A SIMULATION BASED ALGORITHM FOR RAILWAY WAGON SCHEDULING

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

By

S. S. PARADKAR

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY, 1991

CERTIFICATE

[illegible]

It is certified that the work contained in the thesis entitled **"A Simulation Based Algorithm for Railway Wagon Scheduling"**, by **S. S. Paradkar (8911405)**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

(Dr. R.R.K. Sharma)

(Thesis Supervisor)

Industrial & Management Engg.

I.I.T. Kanpur

February, 1991

ABSTRACT

There exists a demand and supply of goods at various stations at different times. There are constraints like capacity of the tracks, number of empty wagons available. Along with the cost of transportation there exists a cost associated with delays. Railways have to route and schedule not only goods loaded wagons but also empty wagons, so as to minimize costs and delays.

A mixed integer linear programming formulation has been given. Literature survey is mainly related to minimum cost flow problems in network and vehicle routing problem. We have developed a heuristic for a single good wagon scheduling problem.

The methodology consists of dividing time horizon in the problem into a number of periods. Now in each time period scheduling of goods wagons and empty wagons is done. In case of goods wagon scheduling we try to solve a min-cost flow problem for a cumulative demand and supply in that period. Then routes are found fitting to the solution of min-cost flow, by simulation. From the scheduling of the goods wagons we come to know about requirements or availability of empty wagons. Allocation of empty wagons, is done by solving transportation problem with a criteria of minimum shortest distance. Simulation used in the routing and scheduling of empty wagons, is based on choosing next available shortest route. Procedure is repeated to cover all the periods in the time horizon.

ACKNOWLEDGMENT

I must give special acknowledgements to Dr. R R K Sharma who has encouraged me to work at my full potential. His guidance has helped me right from the selection of the topic for thesis, upto giving finishing touches to my thesis work. No words will be sufficient for expressing thanks to him.

I am indebted to all teachers of our department. They provided a strong foundation, for doing my thesis work.

I also express my thanks to all my class-mates who were ever ready to help me at any moment. It is impossible for me to forget the friendship with them.

S. S. Paradkar.

TABLE OF CONTENTS

Chapter			Page
1	**Introduction**		1
	1.1	Indian Railways	1
	1.2	Characteristics of the Goods Wagon Scheduling Problem	2
	1.3	The Outline of the Problem	3
	1.4	Organization of the Thesis	3
2	**Formulation of the Problem**		6
	2.1	Introduction	6
	2.2	Formulation	6
	2.2.1	Notations	6
	2.2.2	A Linear Programming Formulation	8
	2.3	Observations from the Formulation	10
3	**Literature Survey**		12
	3.1	Introduction	12
	3.2	Minimum Cost Flow Problem	12
	3.3	Vehicle Routing Problem	14
	3.3.1	Vehicle Routing Problem with Time Windows	14
	3.3.2	Pickup and Delivery Problem	18
	3.3.3	The Multi-Period Vehicle Routing Problem	20

4 Methodology 22

4.1 Introduction 22

4.2 Input 22

4.3 Generate the Information For Given Network 23

4.4 Scheduling of Goods Wagons 24

4.4.1 Solve the Mincost Flow for the Current Period 25

4.4.2 Solving the Tranportation Problem for Goods Wagons 26

4.4.3 Simulating to find the Routes for Goods Wagon Scheduling 29

4.5 Empty Wagon Scheduling 32

4.5.1 Solving the Transportation Problem for Empty Wagons 26

4.5.2 Deciding the routes for empty wagons by simulation 33

4.6 Observation 36

5 Results 37

5.1 Introduction 37

5.2 Numerical Problem 37

5.2.1 Data about arcs of the railway Network 37

5.2.2 Data about the availability of supply 40

5.2.3 Data about the demand requirements 41

5.3 Output 42

5.4 Final Results 42

5.5 Discussion of the results obtained 45

6 Conclusion and Directions for future work 47

6.1 Conclusions 47

6.2 Directions for Future Work 48

References 50

Bibliography 54

Appendix A 55

Appendix B 60

Appendix C 108

LIST OF FIGURES AND TABLES

Figure No.		Page
4.1	Concise Picture of Algorithm	35
5.1	Railway Network Used in the Problem	39
5.2	Total Cost Vs. No. of Wagons	44
5.3	Delay Vs. No. of Empty Wagons available	44
C.1	Cost Vs.Wagons Available problem 3 & 4	109
C.2	Delay Vs. Wagons Available Problem 3 & 4	109
C.3	Cost Vs.Wagons Available problem 5 & 6	112
C.4	Delay Vs. Wagons Available Problem 5 & 6	112
C.5	Cost Vs.Wagons Available problem 7 & 8	115
C.6	Delay Vs. Wagons Available Problem 7 & 8	115

Table No.		
5.1	Arcs of the network	37
5.2	Supply at various nodes	40
5.3	Demand at various nodes	41
5.4	Results, Problem 1	42
5.5	Results, Problem 2	43
C.1	Results, Problem 3 & 4	108
C.2.1	Supply at various nodes, problem 5 & 6	110
C.2.2	Demand at various nodes, problem 5 & 6	110
C.2.3	Capacities of arcs, problem 5 & 6	111
C.2.4	Results, Problem 5	111
C.2.5	Results, Problem 6	111

C.3.1	Supply at various nodes, problem 7 & 8	113
C.3.2	Demand at various nodes, problem 7 & 8	113
C.3.3	Capacities of arcs, problem 7 & 8	114
C.3.4	Results, Problem 7	114
C.3.5	Results, Problem 8	114

CHAPTER 1
INTRODUCTION

1.1 INDIAN RAILWAYS

The first railway started in India in 1853. It ran over a route of 34 km long, from Bombay V.T. to Thane. Since then Indian Railway's network has grown enormously. Today, railways run over a route 70,000 km. long. It is the largest railway network in Asia and also fourth largest in world. Operations of railways have grown tremendously. It has emerged as a largest organization in service sector in India. In the world Indian Railways is second largest railway under a single management. To carry out the operations of railways, it is required to employ 20 lac employees, which is more than any other organization in India. Even though Railways, have enormously grown, both passenger and goods traffic has grown at a much higher rates straining the resources of railways. This has imposed many constraints.

The Indian Railways haul 40 crore tonnes of goods in a year. Mainly it is the freight movement, that contributes to the profit of Railways. In fact Indian Railway bears a loss of more than Rs. 1500 crores on passenger transportation. Hence it would not be wrong to say, that goods transportation is of primary importance to the railways because of the revenue it earns and profits it contributes. So also the scheduling of goods trains is the area of interest of our thesis work.

1.2 CHARACTERISTICS OF THE GOODS SCHEDULING PROBLEM

The passenger trains run according to the fixed time table. Even the new trains which are introduced are scheduled at time mostly according to the convenience of the passenger. Very few trains are introduced every year. As such, doing the scheduling technically, has very little scope. Contrary to this, goods trains do not have fixed time table, not even fixed routes, halts. The railways have to do entire scheduling (both day to day and long run) of goods trains.

The Indian Railways is owned by Government of India. Hence, many a times it has to look after not only the movement of the goods, but also the distribution of the goods. It includes goods like food grain, petroleum products, fertilizers, cement, steel so on. Hence the nature of problem is some what tougher than the case in which customer would like to send goods from one place to another. This is because it is required to consider empty wagon scheduling and also time delays and accompanying cost increases.

1.3 THE OUTLINE OF THE PROBLEM

In a typical transportation problem from operations research, we have warehouses which can supply certain quantity of goods. Also there are markets which have certain amount of demand to be satisfied. The cost of transporting from each warehouse to every market (cost matrix) is known. Similar to transportation problem, we have in our railway network supply nodes (stations) and demand nodes. We know for each arc in the network the cost of transporting both loaded wagon trains and also empty wagon trains. It should be noted that in our case cost of transportation can not be simply specified as cost of transportation from a supply node to demand node. This is because we have to consider not only the cost of transporting loaded goods but also that of empty wagons.

Amongst the many constraints we have the important one being the capacity of the rail tracks, expressed in terms of number of wagons / trains that can be moved over a track in particular direction in unit time. This makes our problem partially is similar to a min-cost flow problem, having many source and sink nodes, instead of just one origin and one destination node. However, the complexity of the real life problem of railways is of orders magnitude more than that of a min-cost flow problem. The railways have to deal with many types of goods which makes problem somewhat similar to the multi commodity min-cost flow problem. Further various types of goods can be transported only in particular type of wagons. Each possible good type and wagon type combination having its own cost of transportation. The most important difficulty lies further when the concept of time is introduced. Every demand has a time deadline, i.e. the latest time by which demand has to be satisfied. So also supplies are available only after certain time and that supplies can be picked up any time after they become available. This concept of time is very important which is normally absent in of the typical transportation problem, travelling salesman problem, min-cost flow problem , vehicle routing problem. In our thesis work, we have dealt with the difficulties faced by the introduction of time associated with demand and supply.

In our context, scheduling involves deciding the number of both goods wagons and empty wagons or trains moving from a origin node to a destination node, including the path along which they move, also deciding the starting time from the origin node, and finding the travelling time along the path, which involves specifying the idle time if any at any of the nodes on the path, on its way to the destination. Giving the delays in reaching the destination if any in reaching the destination.

We use the entire model developed, to analyze how introduction of more resources like wagons can reduce time delays, cost etc.

1.4 ORGANIZATION OF THE THESIS

Rest of the thesis is organized as follows. In the chapter 2, we give mathematical formulation of the problem. Here we have mixed integer linear programming formulation. This formulation reveals the complexity of the problem. In the chapter 3, we have reviewed the literature on related topics. In the chapter 4, we begin by giving the reasons for choosing the particular methodologies, from among the various methodologies. This is followed by complete presentation of the methodology we have adopted. In the chapter 5, we take up a illustrative numerical problem. Then we present the computer results obtained, by running the program. The chapter 6, shows how the model developed can be used for deciding the critical investments like that of wagons. Finally, we present the conclusions derived from our results. Also we have suggested directions for future work in the last section.

CHAPTER 2
FORMULATION OF THE PROBLEM

2.1 INTRODUCTION

In the previous chapter, the outline of the Railway Scheduling Problem was presented. In this chapter we present a mixed integer linear programming formulation of the problem.

There are demands and supplies for different types of goods at different stations on the railway network at various times. We have to schedule the goods wagons and empty wagons so as to satisfy the demands and supplies. Decisions regarding the movement of wagons and the type of goods they carry are to be taken. Loading and unloading schedules of wagons over a period of time are also to be decided.

2.2 FORMULATION

2.2.1 Notations

Decision Variables

WUL_{ikwt}	No. of Wagons of type w that get unloaded at i with goods of type k on day t.
WL_{ikwt}	No. of wagons of type w that get loaded at i with good k on day t.
WLM_{ijkwt}	No. of loaded wagons of type w that start moving from i

	to j carrying goods of type k on day t.
WLI_{ikwt}	No. of loaded wagons of type w that remain idle at i with goods k on day t.
WEM_{ijwt}	No. of empty wagons of type w that start moving from i to j on day t.
WEI_{iwt}	No. of empty wagons of type w that remain idle at i on day t.
DEL_{ikt}	Shortfall at node i, for goods of type k at time t.

Constants (Data)

CL_{ijkw}	Cost of carrying from i to j the goods of type k in the wagons of type w.
CE_{ijw}	Cost of carrying from i to j empty wagons of type w.
t_{ij}	Time (days) required to move wagons from i to j.
U_{ij}	Maximum no. of wagons of any type that can move between i and j on any day.
WAV_{w}	Total no. of wagons of type w that are available at the beginning and also in the entire planning horizon.
D_{ikt}	Demand at node i, for goods of type k, at time t.
S_{ikt}	Supply at node i, for goods of type k, at time t.
C_{kw}	Capacity of wagons of type w to carry goods of type k.
CD_{ik}	Cost of delay at node i, for goods of type k.
I	No. of stations (nodes) on the railway network.
T	Planning horizon in days.
W	Number of different types of wagons.

2.2.2 A Linear Programming Formulation

Objective Function

$$\text{Minimize} \quad \sum_i \sum_j \sum_k \sum_w CL_{ijkw} \cdot (\sum_t WLM_{ijkwt}) + \sum_i \sum_j \sum_w (CE_{ijw} \cdot (\sum_t WEM_{ijwt})$$
$$\sum_i \sum_k CD_{ik} \cdot (\sum_t DEL_{ikt})$$

Constraints

(1) Empty wagons balance

Sum of number of empty wagons unloaded, idle on day t-land arriving on day t, a node i, on a day is equal to the number of wagons loaded, idle, left on day t at node i. Here it is assumed that loading / unloading takes one day. e.g. Wagon unloaded on the day t-1 can not leave on the same day either as empty or as loaded wagon but can leave on next day or later.

$$\sum_k WUL_{ikw(t-1)} + WEI_{iw(t-1)} + \sum_j WEM_{jiw(t-t_{ji})}$$
$$= \sum_k WL_{ikwt} + WEI_{iwt} + \sum_j WEM_{ijwt} \qquad \text{for all } i = 1 .. I, \; w = 1 .. W, \; t = 1 .. T$$

Similarly,

(2) Loaded Wagons Balance

Sum of number of wagons loaded, idle and arriving at a node i, on a day is equal to the number of wagons unloaded, idle and left node i.

$$\sum_k WL_{ikw(t-1)} + \sum_k WLI_{ikw(t-1)} + \sum_j \sum_k WLM_{jikw(t-t_{ji})} \qquad \text{for all } i = 1..I, \; w = 1..W, \; t = 1..T$$
$$= \sum_k WUL_{ikwt} + \sum_k WLI_{ikwt} + \sum_j \sum_k WLM_{ijkwt}$$

(3) Satisfying demand and supply

The cumulative capacity of wagons unloaded for satisfying demand at any node i, should be greater than the demand existing for all the types of goods, at any time.

$$\sum_{t=0}^{t_1} WUL_{ikwt} \cdot C_{kw} + DEL_{ikt} \geq \sum_{t=0}^{t_1} D_{ikt}$$

for demand
for all $t_1 = 1 \ldots T$
$\forall$ i, k.

Similarly, the cumulative capacity of the wagons loaded at a node should not exceed the supplies available at that node i, for all types of goods, at any time.

$$\sum_{t=0}^{t_1} WL_{ikwt} \cdot C_{kw} \leq \sum_{t=0}^{t_1} S_{ikt}$$

for supply
for all $t_1 = 1 \ldots T$
$\forall$ i, k.

(4) Upper bound on wagons in transit

$$\sum_{w} WEM_{ijwt} + \sum_{w} WEM_{jiwt} + \sum_{k}\sum_{w} WLM_{ijkwt} + \sum_{k}\sum_{w} WLM_{jikwt} \leq U_{ij}$$

for all links (joining i&j)

for all t

We assume that loaded wagon which has to be unloaded, can not remain idle at a node (station). It has to be unloaded and kept idle. In other words the loaded wagon can remain idle at a node only if it is to be transported to some other node sooner or later and not if it is to be unloaded. Hence no. of unloaded wagons has to be less than the no. of wagons which arrive at a node for same type of wagon carrying same type of goods

$$WUL_{ikwt} \leq \sum_{j} WLM_{jikw(t-t_{ji})}$$

for all k, w & t

(6) Total no. of wagons in the network is fixed

$$\sum_i \sum_k WLI_{ikwt} + \sum_i WEI_{iwt} = WAV_w \qquad \text{for all } w \text{ and at } t = 0$$

(those at starting)

2.3 OBSERVATIONS FROM THE FORMULATION

In the last section we have given a mixed integer linear programming formulation for the problem of goods and wagon scheduling.

It may be seen that we have included cost for not meeting the demand in time. The costs CD_{ik} could be appropriately chosen to reduce the delays. However, if we try to solve the problem according to above formulation, then we find that even for a small network, the number of variables and number of constraints is very large. For example, if we want to solve a problem which involves a railway network of say 10 nodes and 20 arcs, only 4 types of goods to be transported, in 3 types of wagons and our time horizon is restricted to 100 discrete units of time on which wagon trains can be scheduled. WLM_{ijkwt} alone requires 16,000 variables to be defined, apart from the other decision variables to be defined and constant data to be given. The total number of constraints is around 25,000. This becomes extremely difficulty to solve even if we approximate all the variables as continuous variables rather than integer variables. This is well beyond the limits of the widely available linear programming packages.

Solomon [29] has shown that a problem of finding a feasible solution to a problem of a similar nature (but with lesser complexity) is a NP complete problem. This issue is discussed in the next chapter on literature survey. We limit ourselves to developing a heuristic method (which is computationally viable) which will generate reasonably good schedules. Hence the main aim of our work, is to develop algorithm and write a computer program for the same. The chapter 4 is devoted to explaining the methodology developed.

CHAPTER 3
LITERATURE SURVEY

3.1 INTRODUCTION

We begin the literature survey by studing the mincost flow problem, where the concept of time does not come in. Later, we study the work of various shades of Vehicle Routing Problem.

3.2 MINIMUM COST FLOW PROBLEM (MCFP)

The initial description of the transportation problem is due to Hitchcock (1941). Since then, there have been a number of papers concerning the computational aspects of MCFP. Surveys of earlier work were done by Charnes and Cooper(1951), Ford and Fulkerson [14], Dantzig [9], Busacker and Saaty [4], Hu [18], and Golden and Magnanti [16].

One of the most important and active area of interest today is the basic simplex solution techniques which form the main tool of the work here. The approach is to use a basis tree to implement a procedure analogous to the primal simplex algorithm for the general linear programming problem. Studies have indicated that this is perhaps the most efficient way to solve pure minimum cost flow problems. The early precursors to this work were Dantzig (1954) and Charnes and Cooper (1951) who applied the simplex technique to the transportation problem. In the seventies several authors have utilized the special representation of the basis trees to increase markedly the computational efficiency of the primal simplex techniques. Langley et al.[20] uses the triple label method in a primal simplex algorithm for the capacitated transportation problem. Glover, Karney and Klingman [15] apply the triple label technique to the pure network problem. They have also shown the superiority of the primal simplex procedure for network flow programming problems which was attested by others.

Another approach, which has also been incorporated in this work, as an alternate solution strategy is the incremental flow procedure. An incremental approach is discussed in Ford and Fulkerson [14]. Assad A. A [1] gives comprehensive survey of the literature dealing with multicommodity flow problems. A decomposition and partitioning approach is used to solve the min-cost flow problem using arc-chain formulation of the problem. The arc-chain formulation of the linear minimum cost flow problem suffers from the defect that the chains in the network should be enumerated in advance. These may be enormous in number and thus preclude solving a large network problem. An early paper of Ford and Fulkerson [13] suggested that the chains be generated as neeeded according to a column generation techniques. Tomlin [32] adapts the procedure in the context of the Dantzig - Wolfe decomposition.

3.3 VEHICLE ROUTING PROBLEM

The vehicle routing problem (VRP) involves the design of a set of minimum cost routes, originating and terminating at a central depot, for a fleet of vehicles which services a set of customers with known demands. Each customer is serviced exactly once and furthermore, all the customers must be assigned to vehicles such that the vehicle such that the vehicle capacities are not exceeded. The VRP area has been intensively studied in the literature, seeMagnanti [23]. Christofides [5] has given optimisation algorithms and also heuristics for the basic VRP. WE refer to Solomon and Desrosiers [31] have described various shades of VRP with Time Window Constraints.

3.3.1 Vehicle Routing Problem with Time Windows

Solomon [31] has given mathematical formulation in which VRPTW and many other problem problem variations are derived from pickup and delivery problem (see section 3.3.2). This formulation appears in the Appendix A. In the VRP with time windows (VRPTW), the above issues have to be dealt with under the added complexity of allowable delivery times, or time windows, stemming from the fact that some customers impose service deadlines and earliest service time constraints. In these problems, the spatial aspect of routing is blended with the temporal aspect of scheduling, which must be performed to ensure the satisfaction of the time window constraints. The service of a customer, can begin within the latest time when the customer will permit the start of service. The times at which services begin are decision variables.

It is worth distinguishing now between hard and soft time windows. In the hard time window case, if a vehicle arrives at a customer too early, it will wait. In addition, due dates cannot be violated. In contrast, in the soft time window case, the time window constraint can be violated at a cost, in a manner similar to dualizing constraints in Lagrangian relaxation context.

The vehicle fleet size and mix, may assumed homogeneous. However, many of the methods proposed can be extended to consider a heterogeneous fleet size. Furthermore, the number of vehicles used can be free, i.e., the fleet size is determined simultaneously with the best set of routes and schedules rather than being fixed a priori.

Additional complexities encountered in the VRPTW are length of route constraints arising from depot time window. It may be noted that the vehicle departure times from the depot are decision variables. Furthermore, precedence relationships among certain customer are encountered in some problems, such as those involving both pickup and deliveries.

Most of the effort has been directed at the operational problem of determining the best set of routes and schedules. In the presence of time window, the total routing and scheduling costs include not only the total travel distance and time costs considered for routing problems, but also the cost of waiting time which is incurred when a vehicle arrives too early at a customer location or when the vehicle is loaded or unloaded. Several authors have also have analyzed the strategic question of the optimal fleet size. Keaton [21] dealt with designing optimal railroad operating plans. Various related decisions are modeled as mixed integer linear program. Solutions are obtained using Lagrangian Relaxation and Heuristic approaches.

The VRPTW has emerged as an important area for progress in handling realistic complications and generalizations of the basic routing model (Schrage [27]). In the last few years we have witnessed the development of a fast growing body of research focused on the vehicle routing and scheduling problem structures with time windows. Given the intrinsic difficulty of this problem class, the later work on the generic VRPTW has typically focused on heuristic methods. The findings of Solomon [29] indicate that this problem class is fundamentally more difficult than the VRP. The VRPTW is NP-hard (by reduction from the VRP) and heuristics have so far offered the most promise for solving realistic size problems.

Solomon [29,30] has designed and analyzed a variety of route construction heuristics for the VRPTW. The results of the extensive computational study reported in Solomon [29] indicate that a sequential time-space insertion algorithm proved to be very successful in a number of important VRPTW environments. Very good initial solutions to a large number of 100 customer test problems were obtained. The excellent performance of such heuristics can be explained by realizing that while routing problems seem to be driven by the assignment of customers to vehicles as indicated by the success of the Fisher and Jaikumar [12], generalized assignment heuristic- the sequencing aspect of the problem seems to drive routing problems dominated by time windows. It is this aspect of the problem that insertion approaches capture so well. A very large computational requirement was also reported by Cook and Russell [8], in early work on the VRPTW for a k-optimal improvement heuristic, which was effective in solving an actual problem with a few time constrained customers. Analytical results concerning the behaviour of VRPTW approximation methods are derived through worst-case analysis of heuristics in solomon [29]. For a variety of heuristics, including improvement methods, it is shown that their worst-case behaviour on n-customer problems, is at least order of n for minimizing the number of vehicles used, the total distance travelled and the total schedule time. While heuristics have been found to be very effective and efficient in solving a wide range of practical size VRPTW, few optimal approaches have been reported, see Kolen et al. [22]. The former authors extend the shortest q-path relaxation algorithms of Christofides et al. [6] to the problem with time window. The largest problem solved to optimality involved 4 vehicles servicing 14 customers with tight time windows.

Ferland and Fortin [11] have given a heuristic approach for the vehicle scheduling problem with sliding time windows The advantage has been taken of the fact that the starting time of each task must fall within a given time interval rather than being fixed. The approach consists of identifying pairs of tasks offering good opportunity costs for reducing the overall cost and finding the ways of modifying starting times to allow them to be linked.

3.3.2 Pickup and Delivery Problems (PDPTW)

PDPTW is a generalization of the VRPTW which is concerned with the construction of optimal routes to satisfy transportation requests, each requiring both pick and delivery under precedence, capacity and time window constraints. Note that the VRPTW is the particular case of PDPTW where the destinations are all the common depot.

Most of the literature on pick-up and delivery problems has appeared for the dial-a-ride problem (DARP). It can be seen that the single vehicle DARPTW is a constrained version of the classical travelling salesman problem. Psaraftis [24] presents two dynamic programming algorithms using respectively backward and forward recursions. The objective function minimizes the total customer inconvenience and "maximum position shifts" define time window constraints. These algorithms require $O(n^2 3^n)$ time (where n is the number of customers), a fact which limits the tractable problem size to no more than 8-10 customers.

Sexton and Bodin [28] give mixed integer nonlinear programming formulation for the single vehicle dial-a-ride problem with given maximum delivery times or specified minimum pickup times to minimize customer inconvenience. Benders' decomposition is used to partition the problem into a routing component (a linear program). For the scheduling problem, an optimal algorithm which exploits the network flow structure of its dual is developed. The overall procedure is heuristic due to the difficulty of solving the routing problem exactly. Its computational tractability stems from the non-iterative nature of the scheduling. Successful computational experience on moderately sized real data is reported. Problems with 7 to 20 customers were solved in an average of 18 seconds of UNIVAC 1100/81A CPU time.

Jaw et al. [19] also use a parallel insertion algorithm which appears to be very effective and efficient in minimizing a weighted combination of customer dis-utility and system cost. The authors report computational experience with simulated and real data on the VAX 11/750. The former problems, involving 250 customers and 10-14 vehicles, took around 20 seconds each.

The Bodin and Sexton [3] procedure is a traditional "cluster first, route second" approach. For a fixed fleet size, it partitions the set of requests into vehicle clusters and solves the resulting single vehicle dial-a-ride problems using the heuristic based on Benders decomposition (Sexton and Bodin [28]). Requests are then moved one at a time from one vehicle to another while attempting to reduce total user inconvenience.

3.3.3 The Multi-Period Vehicle Routing Problem (MPVRP)

Here, the time windows are full days and a service activity must occur on a specified number of days of the planning VRPTW where each customer has to be visited a given number of times within its multiple time windows. Most of the existing literature is based on application that require the solution of large scale problems.

Beltrami and Bodin [2] address this problem in the context of routing hoist compactor trucks. The problem, which involves customers requiring service either 3 or 6 times per week, is solved heuristically. One approach involves developing the routes first and then assigning to days of the week. In the second approach, routing is performed after the customers have been randomly pre assigned to days of the week.

Rusell and Igo [26] extened several routing heuristics to assign accounts to days of the week such that the weekly travel distance is minimized. The refuse collection problem considered involves the routing 4 trucks through 490 location requiring service from 1 to 6 times per week. The original problem is transformed into a pure routing problem by creating multiple copies of each account that requires service several times a week, resulting in a 776 customer problem. The best approach found was to assign the locations to days of the week using a simple heuristic, solve the resulting problem, where the spacing conditions are enforced and improve the solution through the use of a k interchange heuristic (M-Tour, Russell [25]). The use of M-Tour proved once again to be computationally prohibitive. Christofides and Beasley [7] develop new heuristics for the MPVRP based on median problem and traveling salesman problem relaxations. The algorithms are characterized by the use of an interchange procedure which improves on initial choices of delivery days combinations for customers.

Haghani A. E. [17] presents the formulation and solution of a combined train routing and makeup and empty car distribution model. The model results in a large scale mixed integer programming problem with nonlinear objective function. A heuristic decomposition technique was developed to solve the model.

CHAPTER 4
METHODOLOGY

4.1 INTRODUCTION

The problem as formulated in chapter 2 is fairly complex and hence we begin solving it by relaxing a class of constraints at a time and solve the relaxed problem by known methods which are efficient. We present the solution method for a single commodity problem, but it can be extended to the multicommodity case.

The detailed steps in the algorithm for solving Railway Wagon Scheduling Problem are as given below.

4.2 INPUT

(i) Network is specified in the form of arcs.

Data about the Arc

(a) Nodes : node1, node2 which are joined by the arc

(b) Distance.

Cost for carrying loaded wagons cost for carrying empty wagons is assumed proportional to distance and both costs are computed.

Cost is expressed as cost/rake for travelling arc. Assumed to be same in both directions.

(ii) Demand / Supply of goods (expressed in wagons (trains))

Input : Node, demand / supply, time .

(iii) Length of time horizon

length of one time period within a time horizon.

We partition the problem into a few distinct time periods say equal to 'n'. This can be done by a suitable heuristic or by simple inspection of data. We have done it by inspection only.

(iv) Availability of empty wagons at various nodes in time period one (or in any other period)

Input : Node, availability of wagons, time.

4.3 GENERATE THE INFORMATION FOR GIVEN NETWORK

For each arc, there is similar arc in reverse direction. Hence ours is undirectional network for which we have following relations holding true.

Number of arcs leaving a node i

= Number of arcs arriving at a node i.

= Degree of a node i / 2

= difference between point[i] and point[i+1].

(1) Arc numbers of all the arcs leaving a particular node i are stored in the form of a forward star network representation.

Arcs leaving node i will be stored in position from point[i] to point[i+1]-1 in the array ftrace[].

(2) Arc numbers of all the arcs arriving a particular node i are stored in the form of a reverse star network representation. Arcs entering node i will be stored in position from point[i] to point[i+1]-1 in the array rtrace[].

4.4 SCHEDULING OF GOODS WAGONS

Out of these 'n' distinct period problem, we pick up the first period problem and totally drop the time constraints on supply and demand. The min-cost flow problem P1 is formulated as below.

Minimize $\sum \sum C_{ij} . X_{ij}$

(1) Flow conservation at all nodes i

Outflow - Inflow = Supply (Demand)

$$\sum_j X_{ij} - \sum_j X_{ji} = B_i \qquad \forall\ i$$

(2) Capacity restriction on flows over the arcs

$$X_{ij} \leq K_{ij} \qquad \forall \text{ arcs}$$

(3) Non negativity of flow constraint

$$X_{ij} \geq 0 \qquad \forall\ (i,j)$$

Where

X_{ij} Flow from node i to node j

C_{ij} Cost of transportation unit quantity from node i to node j

B_i Supply (or Demand) at node i

K_{ij} Capacity of arc from node i to node j

The solution to the above problem gives the flows over the arcs in a period. Various fractions of the flow over the same arc can be a part of the different routes from various supply nodes to different demand nodes. Solve the min-cost flow problem as given below.

4.4.1 Solve the min-cost flow for the current period

Find the cumulative demand / supply of goods within the current time period. This is done because demand / supply may occur at different times in same time period. The demands must be satisfied within the same time period only and also at time mentioned. However supplies not exhausted during past time periods are considered available in the current time period also.

Cumulative demand = $\sum$ Demand for current period.

Cumulative supply = $\sum$ Supply for current period + residual supply of previous periods

With this information about cumulative demand / supply at each node and the data about the network, solve the min-cost flow problem for the current time period. The capacities of the arcs used in solving min-cost flow problem are specified for entire period. The min-cost flow is formulated as a linear program and solved by a revised simplex method.

The output of the min-cost flow problem is in the form of flows over various arcs. The flow may exist in both the directions on a arc (arcs have been separately identified for each direction). Since the individual arc flows have already been decided the cost of transportation is same irrespective of the route chosen and the decisions about which supply node supplies towhich demand node, since it fits in solution given by the min-cost flow.

4.4.2 Solving the Transportation Problem for goods wagons

Take a demand node for all those nodes having a demand and each time it has a demand. Similarly have a supply node for all those nodes having a supply and each time it has a supply. Thus more than one copies of same node will exist if and only if demand / supply exists more than once till end of current time period.

Now we wish to know which node is to supply which demand node and also the likely quantity supplied. While doing so it is desirable to minimize the delays. We solve the following problem P3 which is the transformed transportation problem.

Formulation of transportation problem is given here.

Problem P3

Maximize $\sum \sum S_{ij} X_{ij}$

Subject to

(1) Demand restriction for node i

$$\sum X_{ij} \geq A_i \qquad \forall\ i = 1,2,\ldots,m$$

(2) Supply restriction from node j

$$\sum X_{ij} \leq B_j \qquad \forall\ j = 1,2,\ldots,n$$

(3) Non negativity of flow constraint

$$X_{ij} \geq 0 \qquad \forall\ (i,j)$$

Where S_{ij} is calculated as follows :

Compute a *shortest distance matrix* (d_{ij}) giving shortest distance between each pair of supply node 'i' and demand node 'j'. Convert it into a *shortest time matrix* (t_{ij}).

Compute a *available time matrix* (A_{ij}) giving difference of time at which supply becomes available and the time at which demand needs to be satisfied.

Compute a *slackness matrix*. (S_{ij})

Slackness = Available time - time for travelling shortest distance

$$S_{ij} = A_{ij} - t_{ij}$$

Use the criteria of maximimum of slackness to decide allocation between demand and supply nodes. This criteria is used so as to minimize time infeasibilities in current and future periods. A simple heuristic approach is used to solve this transportation problem.

Solution procedure to solve objective function by maximum slackness criteria is given below.

Heuristic One

Step 1 Find out slackness matrix (also known as transportation table)

Step 2 Select a variable with maximum slackness as basic variable. It is assigned a value equal to minimum of supply or demand. Update the residual demand and supply by subtracting the assigned value.

Step 3 *If* there is tie in the selection of the variable with a maximum slackness, *then* find the penalties for not making that variable as a basic variable. Break the tie by choosing the variable which has maximum penalty.

Step 4 Repeat the procedure till all demand is satisfied.

We try to stick to the above solution of transportation problem P3 and find the routes from supply node to demand node within framework of solution given by min-cost flow.

If we get a infeasible solution to the min-cost flow then we need to solve a different transportation problem which will reasonabally take care of the costs of transportation. We solve a transportation problem P4, which minimizes the sum of the shortest distances.

Formulation of transportation problem which minimizes the sum of shortest distances is given here.

Problem P4

Minimize $\Sigma \Sigma D_{ij} X_{ij}$

Subject to

(1) Demand restriction for node i

$$\Sigma X_{ij} \geq A_i \qquad \forall\ i = 1,2,..,m$$

(2) Supply restriction from node j

$$\Sigma X_{ij} \leq B_j \qquad \forall\ j = 1,2,..,n$$

(3) Non negativity of flow constraint

$$X_{ij} \geq 0 \qquad \forall\ (i,j)$$

Where a *shortest distance matrix* (D_{ij}) giving shortest distance between each pair of supply node 'i' and demand node 'j'.

Use the criteria of minimum of shortest distance to decide allocation between demand and supply nodes. A simple heuristic approach is used to solve this transportation problem.

Solution procedure to solve objective function by minimum shortest distance criteria.

Heuristic Two

Step 1 Find out shortest distance matrix (transportation table)

Step 2 Select a variable with minimum shortest distance as basic variable. It is assigned a value equal to minimum of supply or demand. Update the residual demand and supply by subtracting the assigned value.

Step 3 *If* there is tie in the selection of the variable with a minimum shortest distance, *then* find the penalties for not making that variable as a basic variable. Break the tie by choosing the variable which has maximum penalty.

Step 4 Repeat the procedure till all demand is satisfied.

4.4.3 Simulating to find the routes for goods wagon scheduling

We start from a supply node and travel along any of the arcs leaving the node if that arc has a flow indicated by min-cost solution. We recursively continue search till we reach a demand node which has some positive allocation for that supply node and demand pair. When a path is found successfully, we send a flow along the entire path. Quantity of flow that can be sent is minimum of 1. minimum arc capacity along the path 2. minimum of supply and demand. In case we fail to reach such a demand node, then we backtrack to the previous node along the route so as to try with some other arc leaving the previous node. This backtracking to the previous node along the route may have to be done recursively till we do not have any node left (other than supply node itself) in the route we are trying. In that case we give up the current supply node, which probably may have surplus supplies which may remain even after satisfying all demand. Alternatively its supplies are available too late and can not satisfy demand without delays hence considered later.

The above procedure is repeated so as to find routes from other supply node to demand nodes. If the demand remains unsatisfied even after trying with all the supply nodes *once*, then we reject the tranportation solution which we developed earlier. It may be noted here that transportation solution developed earlier with criteria of maximizing the slackness was just to minimize any time infeasibility that may occur during goods wagon scheduling or consequent empty wagon scheduling. Hence even if we do not stick to the allocation, it is always possible to find routes from the supply node to the demand nodes within the framework of the min-cost solution i.e. without exceeding the flows indicated by min-cost solution which, in turn do not exceed the capacity of the arc specified for the period.

If our problem is too tight problem then the infeasibility is found while solving the min-cost flow itself. In such cases we use minimizing the sum of the shortest distance between the supply and demand node pair as the criteria for deciding the allocations. Now the shortest available routes are found by simulation. Then, this approach becomes very much similar to empty wagon scheduling described in details later on. However in case of goods wagon, their scheduling is always done so as to reach the destination just in time (whenever possible). On the other hand in case of empty wagon, scheduling is done so as to start from the source node as soon as they become available.

When sending the flow along a route found

a. Check time feasibility if any, and find the amount of delays when it occurs.

b. Keep the account of residual capacity of each arc on each day.

Goods Flows are scheduled, so as to reach demand node just in time of need, rather than sending the flow at a time when supplies are available.

This reduces the tightness in scheduling and time infeasibility that might be encountered during empty wagon scheduling. The exception to this scheduling rule, is the situation when supplies are not available within required time. Then the goods trains are scheduled to leave the supply node only when supplies become available, even though they will reach demand node with delay.

4.5 EMPTY WAGON SCHEDULING

The demand and supply of goods (goods trains) acts as a source of supply and demand for empty wagons respectively. However the time of demand / supply need not be same as that in data. Delays might have occurred while satisfying demand of goods (which leads to time infeasibility). It is possible to have a situation in which demand gets partially satisfied in time and partially with delays. Also supply of goods may be picked up at a later time, rather than the time at which it becomes available. So also a smaller portion of the total supplies available at a node may be picked up at different times.

Thus to get precise information for requirement and availability of empty wagons, it is necessary to keep account of (1) Number of empty wagons and (2) time at which they are required or are available as soon as loading or unloading of goods takes place during goods wagon scheduling.

4.5.1 Solving the tranportation problem for Empty wagons

Similar to goods wagon scheduling we shall form demand nodes and supply nodes for empty wagons each time there is requirement for or availability of empty wagons. Then we form a transportation problem to decide which supply node will supply which demand node. However unlike previous case the basis for allocations is simply shortest distance matrix which gives the shortest distance between every pair of supply and demand node.

The reason why we have used criteria of maximizing the slackness while goods wagon scheduling and that of minimizing the sum of shortest distance while finding allocations during empty wagon scheduling has to be noted. During goods wagon scheduling min-cost flow was solved first and the purpose of solving the transportation problem with maximum slackness was to minimize the time infeasibilities that may occur during goods wagon and empty scheduling. Overall cost factor was already taken care of when we solved min-cost flow problem. The reason why shortest distance matrix is used as a basis for allocations during empty wagon scheduling is that we would like to minimize the costs of transportation for empty wagon scheduling also. We have also found that maximizing the slackness is a poor criteria during empty scheduling. Also it is not needed because nothing further depends on empty wagon scheduling, unlike the case of goods wagon scheduling from which empty wagon scheduling is derived.

4.5.2 Deciding the routes for empty wagons by simulation

Procedure for deciding the routes from supply node to demand node for empty wagons is somewhat simpler in case of empty wagon scheduling than goods wagon scheduling. Here we strictly stick to the allocations. Moreover unlike the min-cost flow of the goods problem, the capacities of arc are not expressed for whole period but for a each unit time within a period, which is a fraction (to be specified) of the capacity for the period.

We take a supply node. We try to send empty wagons demand node along a shortest path. In case, we find that there no capacity available on a particular arc in the path on particular day then

(1) We try, keeping the empty wagons idle just before the arc which there is insufficient capacity available on that particul day till, sufficient capacity becomes available.

(2) In case we find that by keeping the empty wagons idle, be that particular arc, it is not possible to reach the destinat in time, then we try to find a next shortest path excluding arc(s) whose capacity has saturated. In case we encounter simil problem in the next shortest path also then we repeat the proc dure. If the distance involved in travelling by next shorte available path is such that it will not be possible to rea destination then we accept that as time infeasibility. wagons are scheduled along a first shortest path only. We note delays that are bound to occur when we send the goods by shortest path only.

The steps 4.4 and 4.5 are repeated for all the time periods in time horizon. The concise picture of the algorithm appears figure 4.1

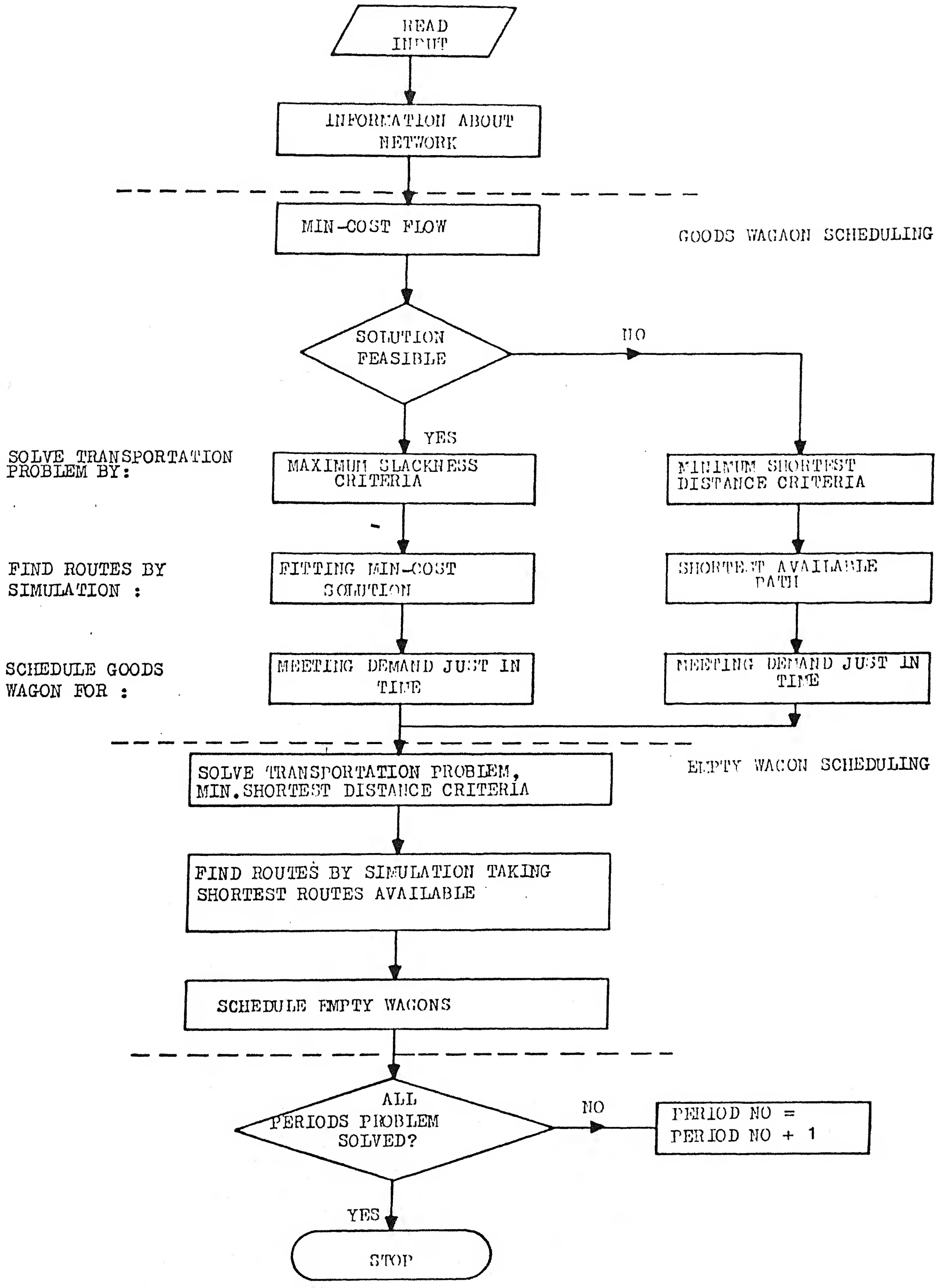

READ INPUT
INFORMATION ABOUT NETWORK
MIN-COST FLOW
GOODS WAGAON SCHEDULING
SOLUTION FEASIBLE
NO
YES
SOLVE TRANSPORTATION PROBLEM BY:
MAXIMUM SLACKNESS CRITERIA
MINIMUM SHORTEST DISTANCE CRITERIA
FIND ROUTES BY SIMULATION :
FITTING MIN-COST SOLUTION
SHORTEST AVAILABLE PATH
SCHEDULE GOODS WAGON FOR :
MEETING DEMAND JUST IN TIME
MEETING DEMAND JUST IN TIME
SOLVE TRANSPORTATION PROBLEM, MIN.SHORTEST DISTANCE CRITERIA
EMPTY WAGON SCHEDULING
FIND ROUTES BY SIMULATION TAKING SHORTEST ROUTES AVAILABLE
SCHEDULE EMPTY WAGONS
ALL PERIODS PROBLEM SOLVED?
NO
PERIOD NO = PERIOD NO + 1
YES
STOP

FIG. 4.1 CONCISE PICTURE OF ALGORITHM

4.6 OBSERVATIONS

The choice of length of the time period seems to be important. If the period length is small then the number of periods in the horizon increases, and there is corresponding increase in the computational time. If the period length is taken to be very large then the wagons remain idle in a period. This is because wagons are scheduled only twice within a time period. Once goods carrying wagons are scheduled and once empty wagons are scheduled, so as to meet the requirements for the goods wagon. Hence there is poor utilization of the wagons when period length is large. Also it may result in quite suboptimal and at times infeasible solutions. In which case we have try again with a smaller length of the period.

CHAPTER 5
RESULTS

5.1 INTRODUCTION

In the previous chapter, we have presented the solution methodology explaining the details of various steps involved in the algorithm coded. The program code has been attached in the appendix B. In this chapter we take up a numerical example. In the section 5.2, the data of the problem is given. Figure 5.1 shows the railway network which is used for the problem. In the section 5.3, the output results have been tabulated. In the last section, we have discussion on the results we have obtained.

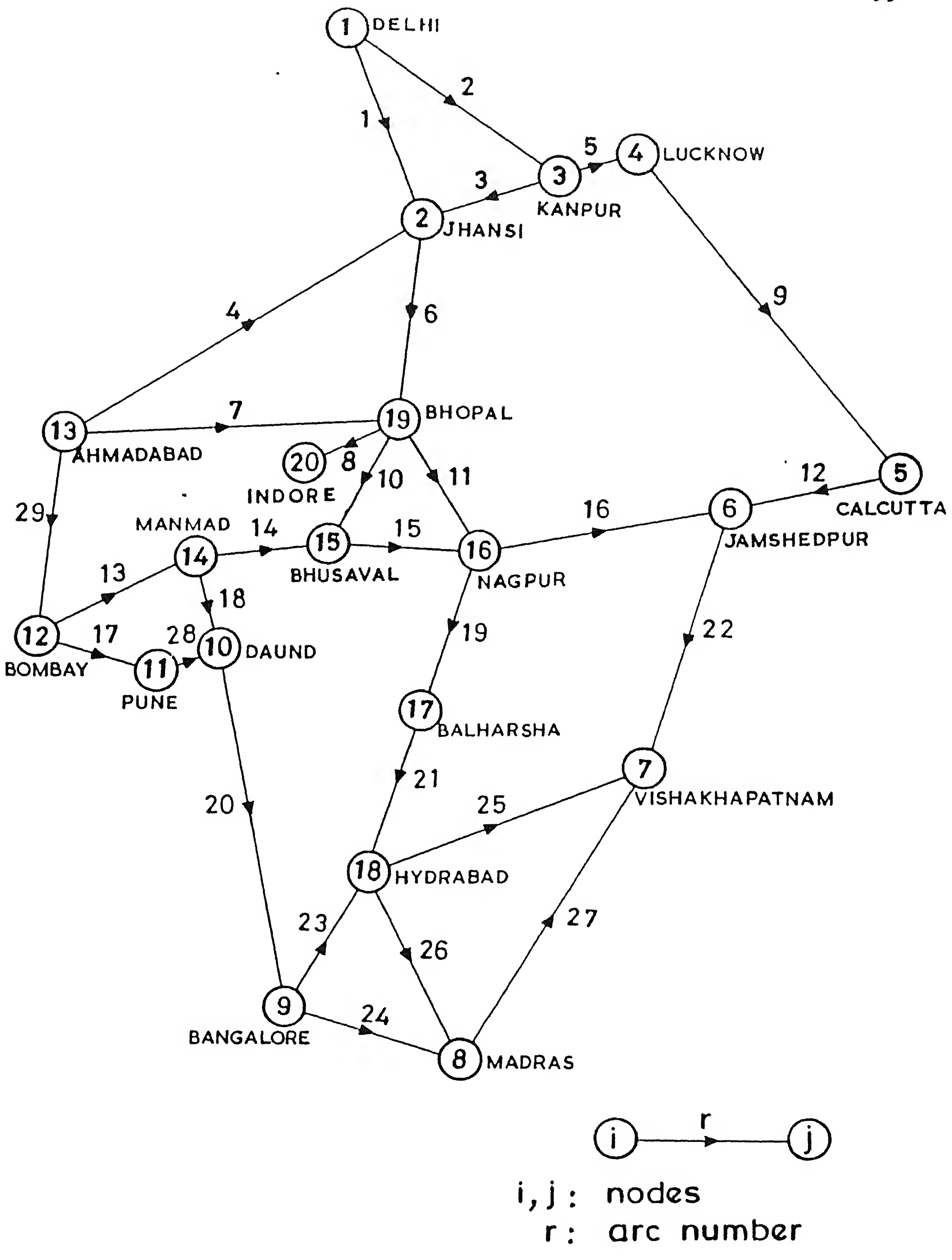
5.2 NUMERICAL PROBLEM

5.2.1 Data about arcs of the railway network

Table 5.1 Arcs of the network

Arc No.	Node1	Node2	Distance (km)	Capacity Problems 1	2
1	Delhi (1)	Jhansi (2)	400	16	26
2	Delhi (1)	Kanpur (3)	400	20	25
3	Kanpur (3)	Jhansi (4)	200	5	5
4	Ahmad.(13)	Jhansi (2)	800	5	5
5	Kanpur (3)	Lucknow (4)	100	25	35

6	Jhansi (2)	Bhopal (19)	300	30	35
7	Ahmad.(13)	Bhopal (19)	550	6	6
8	Bhopal (19)	Indore (20)	200	25	25
9	Lucknow (4)	Calcutta (5)	1000	13	25
10	Bhopal (19)	Bhusaval(15)	400	7	8
11	Bhopal (19)	Nagpur (16)	350	9	7
12	Calcut. (5)	Jamshed.(6)	250	13	25
13	Bombay (12)	Manmad (14)	250	25	19
14	Manmad (14)	Bhusav. (15)	200	16	16
15	Bhusa. (15)	Nagpur (16)	400	7	7
16	Nagpur (16)	Jamshed. (6)	250	25	23
17	Bombay (12)	Pune (11)	200	6	6
18	Manmad (14)	Daund (10)	250	7	7
19	Nagpur (16)	Balhar. (17)	200	19	16
20	Daund (10)	Bangal. (9)	900	6	6
21	Balhar.(17)	Hydrab. (18)	350	9	9
22	Jamshed.(6)	Vishakh. (7)	800	6	6
23	Bangal.(9)	Hydrab. (18)	700	6	6
24	Bangal.(9)	Madras (8)	350	25	23
25	Hydrab.(18)	Vishakh. (7)	600	25	21
26	Hydrab.(18)	Madras (8)	700	5	5
27	Madras (8)	Vishakh. (7)	850	5	5
28	Pune (11)	Daund (10)	100	5	5
29	Ahmad. (13)	Bombay (12)	500	25	25



RAILWAY NETWORK USED IN THE PROBLEM.

Fig. 5.1

Network shown contains 20 nodes (n) and 29 arcs (m). There are also 29 more arcs in direction just opposite to the one shown in the network. These are identified by numbers from m+1 to 2m. The cost of the wagon movement along a arc is taken proportional to the distance.

The cost of loaded wagon train movement = distance (km) / 10.

The cost of empty wagon train movement = distance (km) /15.

Also to have the consistancy in the unitsfollowing conversions is used.

1 unit time = 4 hrs = 200 km.

1 day = 6 units of time.

5.2.2 Data about the availability of supply

Table 5.2 Supply at various nodes

Supply node	Supply, Time(day)
2 Jhansi	5,2; 10,3; 15,5; 4,6; 6,7; 7,8; 10,10; 5,12; 18,13
18 Hydrabad	10,2; 5,4; 18,5; 10,6; 5,8; 18,9; 4,10; 6,11; 7,12
4 Lucknow	2,1; 5,4; 8,5; 10,6; 12,9; 2,10; 5,12; 8,13
9 Bangalore	4,1; 6,2; 7,3; 8,6; 5,8; 10,10; 10,11; 10,12; 15,13
6 Jamshed.	8,2; 5,4; 10,6; 10,7; 10,8; 15,9; 8,10; 5,12
1 Delhi	10,1; 12,4; 2,6; 5,8; 8,9; 10,10; 12,13
12 Bombay	10,1; 10,2; 10,4; 15,5; 5,6; 10,7; 15,9; 5,10; 10,11; 15,13

5.2.3 Data about the demand requirements

Table 5.3 Demand at various nodes

Demand node	Demand, time (day)
19 Bhopal	10,5; 15,6; 10,9; 5,10; 4,11; 8,13; 7,14
13 Ahmadabad	2,6; 5,7; 10,9; 5,11; 8,13; 7,15
7 Vijaywada	20,6; 10,8; 5,9; 15,12; 15,13; 10,14
8 Madras	15,7; 10,8; 20,9; 10,11; 10,13; 15,14
14 Manmad	5,4; 15,8; 2,9; 5,10; 2,13; 5,14
16 Nagpur	8,5; 7,6; 8,9; 7,10; 5,13; 10,16
5 Calcutta	10,4; 5,6; 5,9; 10,12; 5,13; 15,16
20 Indore	5,4; 10,7; 15,9; 10,10; 10,13; 5,14; 4,16
3 Kanpur	10,4; 5,5; 4,6; 10,9; 15,10; 10,13; 5,15

The empty wagons made available in the entire network in the second time period is varied for both problems. Solution is obtained for four cases of each problem.

1. 175 2. 200 3. 225 4. 250

5.3 OUTPUT

The output solution is volumnious. We get following different outputs.

1. Distribution of number of trains delayed vs. amount of delay

   Percentage frequency of delays vs. amount of delays.

   Percentage of delay vs. amount of delays

   Average delay.

2. Routes followed by the trains and schedule of the trains seperately for both goods movement and empty wagon movement. It is expressed as nodes on the path and the starting time from each node on the path which is stated taking into consideration idle time if any at a node.
3. Load on each arc, each day.
   The maximum load on the arc, and the time at which it occurs
4. The cost of scheduling for both goods trains and empty trains in each time period. And total cost of the schedule generated.

5.4 FINAL RESULTS

Table 5.4

Problem 1

Number of Empty Wagons	Total Cost of Scheduling	Average Delays
175	35,886	5.13
200	35,613	5.23
225	35,096	4.90
250	33,937	4.53

Table 5.5

Problem2

Number of Empty Wagons	Total Cost of Scheduling	Average Delays
175	38,786	3.49
200	38,416	3.37
225	38,020	3.30
250	36,706	3.01

This results are plotted in the figure 5.2 and figure 5.3

Total Cost Vs. No. of Wagons Available

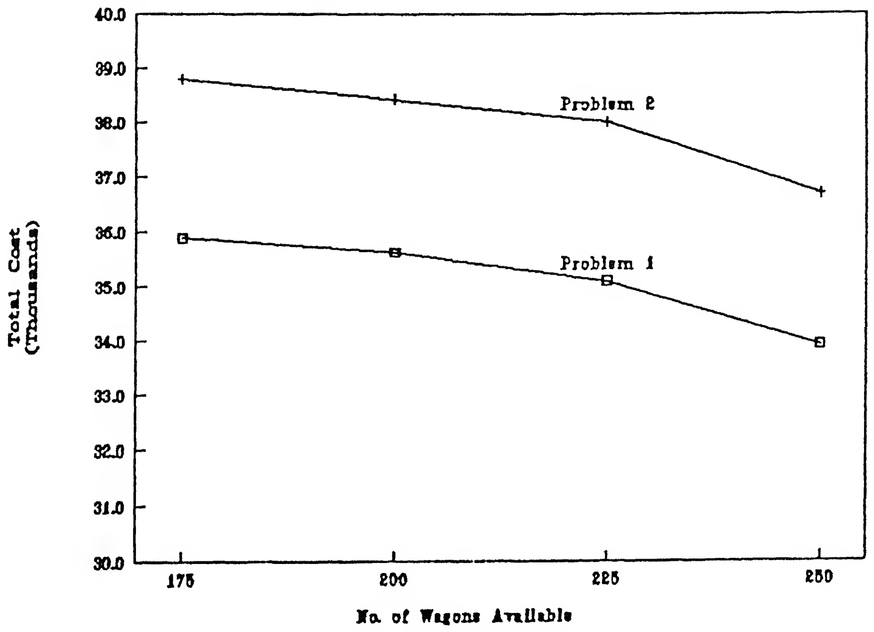


FIG. 5.2

Delay Vs. No. of Empty Wagons Available

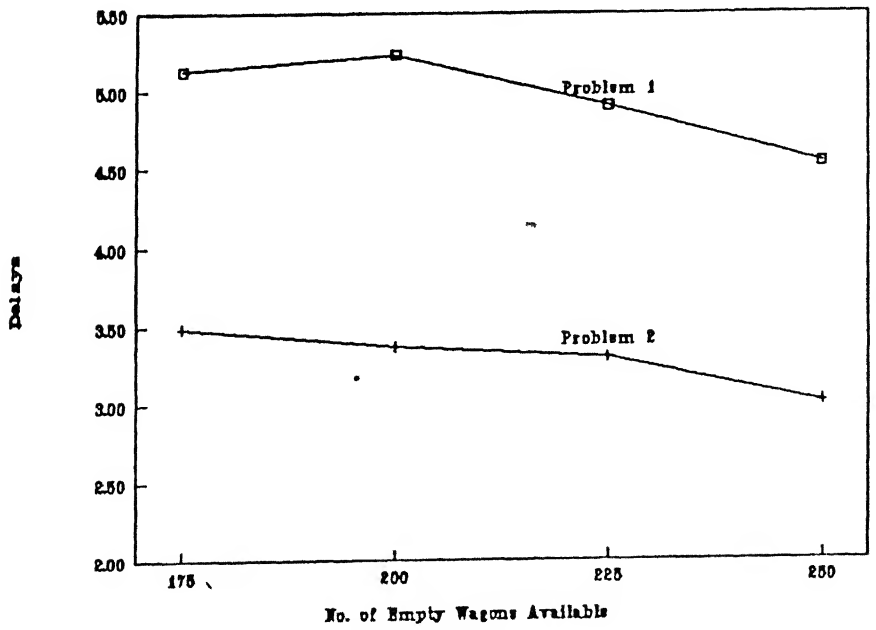


FIG. 5.3

5.5 DISCUSSION OF THE RESULTS OBTAINED

Fig. 5.2 gives a graph of total cost of transportation vs. number of empty wagons available. One can very easily notice that the total cost of the transportation decreases with the increase in the number of empty wagons available. It is simple to interpret it. When the number of empty wagons available is inadequate, then these need fetched from far off places before these can be used for goods transportation, which increases the total cost of the transportation.

Figure 5.3 is a graph of Delays in the movement of the wagons. In this graph also it can observed that there is decrease in delays with increase in the availability of the wagons. Very similar explanation seems to be valid for this behaviour also. i.e. more the number of empty wagons available, lesser will be the need to bring them from over a long distance. However a tradeoff is possible for the decision maker between higher cost of more trains and the lesser delays and transportation costs.

If we compare the results obtained for two problems then we find that the problem 2 in which we incur more total cost of transportation has correspondingly lesser amount of delays taking place and vice versa. However in problem 1 we find that as the number of empty wagons available is decreased from 200 to 175 there is a slight decrease in the delays (even though the cost of transportation has increased). This slight discrepancy probably due to the fact that our heuristic is "single pass" in nature.

CHAPTER 6
CONCLUSION
AND
DIRECTIONS FOR FUTURE WORK

6.1 CONCLUSIONS

The computational experience with the algorithm that has been developed seems to be reasonably well. The moderately sized problem, has been solved on HP 9000, in approximately 30 secs of the CPU time.

We feel that the graphs that are obtained, can be very useful for Railways for taking decisions regarding the investments to be made in the critical assets like wagons. In reality usually there is some costs associated with the delays e.g. Overtime paid to the employees, more fuel and other operating costs for engines. If the cost of delays can be quantified, then it should be possible to find the sum of cost of transportation and cost of delays. It will help Railways for getting a idea of approximate investments desired so as to have the minimum net cost of transportation.

6.2 DIRECTIONS FOR FUTURE WORK

After working on the problem of Railway Wagon Scheduling we have realized various dimensions of this area. There is scope for continuing work on this problem area.

It will be useful to work in this area to develop a methodology for dealing with multicommodity problem. It may be enough to deal with few goods. This is because 80% of the total freight movement is due to few types of goods like steel, food grain, fertilizers, cement. Attempt could be made by modifying methodology so as to incorporate multicommodity min-cost flow and multidimensional assignment problem.

Given the difficult nature of the problem it is felt that heuristics need to be developed. Also heuristic must be of the "multipass" type. So that each time we are able to improve on the solution of the previous iterations.

The scope of the problem can be widened so as to a decision support system for other important investment areas of Railways e.g. Rails of the Railway lines. Certain rail routes are heavily loaded, where as others are scarcely utilized. The study of this aspect will be useful in taking decisions regarding increasing the capacity of the routes.

It will be desirable to take a real life data. Also, an attempt needs to be made to solve at least one or two problems optimally. It will give us the yardstick to compare the performance of the heuristic approaches developed for such problem.

REFERENCES

1. Assad, A.A, "Multicommodity Network Flows - A Survey," *Networks* 8, 37-92 (1978)
2. Beltrami, E. and L. Bodin, " Networks and Vehicle Routing for Municipal Waste Collection," *Networks* 4, 65-94 (1974)
3. Bodin, L. and T. Sexton, "The Multi-Vehicle Subscriber Dial-a-Ride Problem," *Management Science* 22, 73-86 (1986)
4. Busacker, R.G. and T.Saaty, *Finite Graphs and Networks*, McGraw-Hill, New York (1965)
5. Christofides, N., "Vehicle Routing," *The Travelling Salesman Problem*, Edited by E.L.Lawler, J.K.Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, John Wiley & Sons Ltd. 431-465 (1985)
6. Christofides, N., A. Mingozzi and P. Toth, "Exact Algorithms for the VRP, Based on Spanning Tree and Shortest Path Relaxations," *Mathematical Programming* 20, 255-282 (1981)
7. Christofides, N., and J. Beasely, "The Period Routing Problem," *Networks* 14, 237-256, (1984)
8. Cook, T.and R. Russell, "A Simulation and Statistical Analysis of Stochastic Vehicle Routing with Timing Constraints," *Decision Science* 9, 673-687 (1978)
9. Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., (1963)
10. Dantzig, G.B., "Application of the Simplex method to a

Transportation Problem," in *Activity Analysis of Production and Allocation*, Edited by T.C.Koopmans, John Wiley & sons, New York (1951)

11. Ferland, J.A., and Luc Fortin, "Vehicle Scheduling with Sliding Time Windows," *European Journal of Operations Research* 38, 213-226 (1989)

12. Fisher, M. and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks* **11**, 109-124 (1981)

13. Ford, L.R. and D.R. Fulkerson, "A Suggested Computation for Maximal Multicommodity Network Flows," *Management Science* **5**, 97-101, (1958)

14. Ford, L.R. and D.R.Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, N.J. (1962)

15. Glover,F., D.Karney and D.Klingman, "Implementation Computational Comparison of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," *Networks* **4**, 191-212 (1974)

16. Golden, B.L. and T.L. Magnanti, "Deterministic Network optimisation : A Bibiligraphy," *Networks* 7, 149-183 (1977)

17. Haghani, A.E.,"Formulation and Solution of a Combined Train Routing and Makeup, and Empty Car Distribution Model," *Transportation Research* B, 23B, No.6. 433-452 (1989)

18. Hu, T.C., *Integer Programming and Network Flows*, Addison-Wesley, Reading Massachusetts (1969)

19. Jaw, J., A. Odoni, H. Psaraftis and N. Wilson, " A Heuristic Algorithm for the Multi-Vehicle Advanced-Request Dial-A-Ride Problem with Time Windows," *Transportation Research* **20B**, 243-257 (1986)

20. Langley, R.W., J.L. Kennington and C.M.Shetty, "Efficient Computational Devices for the Capacitated Transportation Problem," *Naval Research Logistics Quarterly* **21**, 637-647 (1974)

21. Keaton, Mark H., "Designing Optimal Railroad Operating Plans : Lagrangian Relaxation and Heuristic Approaches," *Transportation Research* B, **23B**, No.6, 415-431 (1989)

22. Kolen, A.W.J., A.H.G. Rinnooy Kan and H.W.J.M. Trienekens, "Vehicle Routing with Time Windows," *Operations Research* **35**, No.2, 266-273 (1987)

23. Magnanti, T., "Combinatorial Optimization and Vehicle Fleet Planning : Perspectives and Prospects," *Networks* **11**, 179-214 (1981)

24. Psaraftis H., "An Exact Algorithm for the Single Vehicle, Many to Many, Immediate Request Dial-A-Ride Problem," *Transportation Science* **14**, 130-154 (1980)

25. Russell, R., "An Effective Heuristic for the M-Tour Travelling Salesman Problem with Some Side Conditions," *Operations Research* **25**, 517-524 (1977)

26. Russell, R. and W. Igo, " An Assignment Routing Problem," *Networks* **9**, 1-17 (1979)

27. Schrage, L., "Formulation and Structure of More Complex / Realistic Routing and Scheduling Problems," *Networks* 11, 229-232 (1981)

28. Sexton, T. and L. Bodin, "Optimzing single Vehicle Many-to-Many Operations with Desired Delivery Times : I. Scheduling II. Routing," *Transportation Science* **19**, 378-410, 411-435 (1985)

29. Solomon, M.M., "On the Worst-Case Performance of Some Heuristics for the Vehicle routing and Scheduling Problem With Time Window Constraints," *Networks* **16**, 161-174 (1986)

30. Solomon, M.M, "Algorithms for Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research* **35**, No.2, 254-265 (1987)

31. Solomon, M.M. and J.Desrosiers, Survey Paper "Time Window Constrained Routing and Scheduling Problems," *Transportation Science* **22**, No.1, 1-13 (1988)

32. Tomlin, J.A., "Minimum-Cost Multicommodity Network Flows," *Operations Research* **14**, 45-51 (1966)

Bibliography

Charnes A. and W.W.Cooper, "The Stepping Stone Method of Explaining Linear Programming Calculations in Transportation Problem," *Management Science*, 1, 49-69 (1954)

Glover, F., D.Karney and D.Klingman, "The Augmented Predecessor Index Method for Locating Stepping-Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Science*, 6, 171-180 (1972)

Hitchcock, F.L. "The Distribution of a Product from Several Sources to Numerous Localities," *Journal of Mathematics and Physics* 20, 224-230 (1941)

Orden, A., "The Transshipment Problem," *Management Science* 2, 276-285 (1956)

APPENDIX A

Here we give a formulation for pick and delivery problem with time windows from Solomon [31]. Then this model is used to develop a taxonomy of routing and scheduling problems with time windows.

Let there be n customers indexed by i. Associate to the pickup location of customer i a node i and to his delivery location a node n+i. Also associate to the depot node 0 and 2n + 1. This creates a clear distinction between customers, their associated locations and the nodes of the network. Note, however, that different nodes may correspond to the same physical location. Therefore, $N = \{1,2,..,n, n+1,n+2,..,2n+1\}$ is the node set for our network, and $P = P^{+} \cup P^{-}$, $P^{+} = \{1,2,..,n\}$, $P^{-} = \{n+1, n+2,..,2n\}$ is the set of nodes other than the depot nodes. We will index both N and P by u and w. Customer i demands that d_i units be shipped from node i to node n+i. Next, let $[a_i, b_i]$ denote the pickup tie window for customer i and let $[a_{n+i}, b_{n+i}]$ denote his delivery time window. Let $V = \{1,2,...,|V|\}$ be the set of vehicles to be routed and scheduled; index this set by v. Let D be the capacity of each vehicle. Let also $[a_o, b_o]$ denote the vehicles' dparture time window from the depot and $[a_{2n+1}, b_{2n+1}]$ their time window for arrival back at the depot.

For each distinct u,w in N, let t_{uv} and c_{uv} represent the travel time and the travel cost form u to w, respectively. Retain only those arcs (u,w) for which (1) $a_u + s_u + t_{uv} \leq b_v$, where s_u is the service time at customer u, and either (2) $d_u + d_v \leq D$, u,w $\in P$, u $\neq$ w, or (2') $d_{u-n} + d_{v-n} \leq D$, u,w $\in P^-$, u $\neq$ w. Finally, let K be the fixed cost associated with each vehicle, incurred if this vehicle is utilized.

Three types of variables are used in the mathematical formulation : binary flow variables X^v_{uv}, u, w $\in$ N, v $\in$ V, u $\neq$ w, time variables T_u, T^1_o, T^v_{2n+1}, u $\in$ P, v $\in$ V, and load variables Y_u, u $\in$ P. Let the binary decision variable X^v_{uv} equal to one if vehicle v travels from node u to node w, and equal to zero otherwise, v $\in$ V, and u, w $\in$ N, u $\neq$ w. Let Tu be the time at which service at customer u begins, u $\in$ P. Let also T^v_o be the time at which vehicle v leaves the depot and T^v_{2n+1} be the time at which it returns to the depot, v $\in$ V. Next, let Y_u be the total load on the vehicle just after it leaves node u, u $\in$ P. It is assumed that the vehicles depart empty from the depot, i.e., $Y_o = 0$. These are decision variables with nonnegativity constraints.

Finally, let $f_u(T_u)$ denote the penalty associated with beginning service at node u at time T_u, u $\in$ P, and $g^v(T^v_{2n+1} - T^v_o)$ be the penalty associated with a route of the indicated duration when traversed by vehicle v $\in$ V. These penalties must be expressed in the same units as the costs c_{uv}. We are now in a position to present the pickup and delivery problem with time windows formally. The mathematical formulation is :

$$\text{Min } \sum_{v\in V}\sum_{u\in N}\sum_{v\in N} c_{uv}X^{v}_{uv} + \sum_{u\in P} fu(Tu) + \sum_{v\in V} g^{v}(T^{v}_{2n+1} - T^{v}_{o}) \qquad (1)$$

subject to

$$\sum_{u\in V}\sum_{v\in N} Xuv = 1, \qquad u \in P^{+} \qquad (2)$$

$$\sum_{v\in N} X_{uv} - \sum_{v\in N} X^{v}_{vu} = 0, \qquad u \in P,\quad v \in V \qquad (3)$$

$$\sum_{v\in P^{+}} X^{v}_{ov} = 1, \qquad v \in V \qquad (4)$$

$$\sum_{u\in P^{-}} X^{v}_{u,2n+1} = 1, \qquad v \in V \qquad (5)$$

$$\sum_{v\in N} X^{v}_{uv} - \sum_{v\in N} X^{v}_{v,n+u} = 0, \qquad u \in P^{+},\quad v \in V \qquad (6)$$

$$T_{u} + s_{u} + t_{u,n+u} \leq T_{n+u}, \qquad u \in P^{+} \qquad (7)$$

$$X^{v}_{uv} = 1 \Rightarrow T_{u} + s_{u} + t_{uv} \leq T_{v}, \qquad u,\ w \in P,\quad v \in V \qquad (8)$$

$$X^{v}_{ov} = 1 \Rightarrow T^{v}_{o} + t_{ov} \leq T_{v}, \qquad u \in P^{+},\quad v \in V \qquad (9)$$

$$X^{v}_{u,2n+1} = 1 \Rightarrow T^{v}_{o} + s_{u} + t_{u,2n+1} \leq T^{v}_{2n+1}, \qquad u \in P^{-},\quad v \in V \qquad (10)$$

$$a_{u} \leq T_{u} \leq b_{u}, \qquad u \in P \qquad (11)$$

$$a_{o} \leq T^{v}_{o} \leq b_{o}, \qquad v \in V \qquad (12)$$

$$a_{2n+1} \leq T^{v}_{2n+1} \leq b_{2n+1}, \qquad v \in V \qquad (13)$$

$$X^{v}_{uv} = 1 \Rightarrow Y_{u} + d_{v} = Y_{v}, \qquad u \in P,\quad w \in P^{+},\quad v \in V \qquad (14)$$

$X^v_{uv} = 1 \Rightarrow Y_u - d_v = Y_v$, $\quad u \in P$, $w \in P^-$, $v \in V$ (15)

$X^v_{ov} = 1 \Rightarrow Y_o + d_v = Y_v$, $\quad w \in P^+$, $v \in V$ (16)

$Y_o = 0$, $0 \le Y \le D$, $\quad u \in P^+$ (17)

X^v_{uv} binary, $\quad u, w \in N$, $v \in V$ (18)

We seek to minimize the sum of the total cost, the total penalty associated with servicing customers too early or too late, and the total penalty associated with routes exceeding a given duration. Constraints (2)-(5) and (18) forma multicommodity min-cost flow problem. Constraints (6) ensure that the same vehicle, v, visits both u and n+u. Constraints (7) are precedence constraints which force node u to be visited befor node n + u. Next, constraints (8)-(10) describe the compatibility requirements between routes and schedules, while constraints (11)-(13) are the time window constraints. Finally, constraints (14)-(16) express the compatibility requirements between routes and vehicle loads, while constraints (17) are the capacity constraints.

Note that the formulation includes a route duration restriction ($b_{2n+1} - a_o$). Note also that the formulation does not include subtour elimination.

Let us now examin how models for different problem variants can be obtained from this formulation. The above forulation is the well known multivehicle, many-to-many dial-a-ride problem with time windows. If we let $|V| = 1$ and eliminate constraint (6) we obtain the single vehicle variant.

Redefine now $N = P \cup \{0,n+1\}$, $P = P^{+} = P^{-} = \{1,2,...,n\}$. To obtain the VRPTW we delete constraints (6), (7) and (15). The constraints (8), as well as (14) ensure subtour elimination. If the VRPTW involves only pickup, define $d_{w} \geq 0$. For delivery VRPTW, $d_{w} \leq 0$, and in constraint (17), set $Yo = D$ and eliminate $Y_{u} \leq D$. The multiperiod VRP can be viewed as VRPTW where the customers are replaced with request for service and for each request there is a time window.

Furthermore, different schortest path problems with time window can be obtained from the above formulation. For example, constraints (1), (3)-(5), (8)-(13), (18) and $|V| = 1$ constitue one shortest path problems with capacity, precedence and time window, can be obtained similarly.

APPENDIX B

PROGRAM FOR RAILWAY WAGON SCHEDULING

```
PROGRAM heuri10 (INPUT, OUTPUT);
        { Trying to determine costs.  Used for solving problems
        with restricted capacity of arcs, so as to make mincost infi}
LABEL   25,50,75,100,200,250,300, 350,375,380,550,575,580,600;
CONST   mmax=100; nmax=200;  nodmax=75; arcmax=150;
        T_hormax = 120; { Max Time horizon (unit time) }
TYPE
      { mainly used for the proce simplex }
        matrix = ARRAY [1..mmax] OF REAL;
        matrint= ARRAY [1..mmax] OF INTEGER;
        row_matrix = ARRAY [1..nmax] OF REAL;
        optimisationtype = ( maximisation, minimisation);
        bigmatrix = ARRAY [1..mmax,1..nmax] OF REAL;
        inequalarray = ARRAY [1..mmax] OF CHAR;
      { mainly used for defining node's data }
        nodetype = 0..nodmax;
        nodeset = SET OF nodetype;
        nodrec = RECORD
                        dema_supl : ARRAY [1..T_hormax] OF REAL;
                        cum_dem_sup_g : REAL;
                        requ_avai : ARRAY [0..T_hormax] OF REAL;
                        cum_dem_sup_w : REAL;
                END;
        nodaray = ARRAY [1..nodmax] OF nodrec;
      { mainly used for the defining arc's  data }
        arc_rec = RECORD
                        node1, node2 : nodetype;
                        dist,cost_g, cost_w,
                        capa,  tmp_capa : REAL;
                        deleted,enter : BOOLEAN;
                        res_cap,load : ARRAY [1..T_hormax] OF REAL;
                        tim_st,tim_dt : REAL;
                END;
        arcaray = ARRAY [1..arcmax] OF arc_rec;
        { used for procedure find_assignment }
          dem_nod = RECORD
                            nd : nodetype;
                            dem : REAL;
                            tim : 1..T_hormax;
                    END;
          ass_dem_nd = ARRAY [1..nodmax] OF dem_nod;
          sup_nod = RECORD
                            nd : nodetype;
                            sup : REAL;
                            tim : 1..T_hormax;
                    END;
          ass_sup_nd = ARRAY [1..nodmax] OF sup_nod;
          sl_ass_typ = ARRAY [1..nodmax,1..nodmax] OF REAL;
          arcset = SET OF INTEGER;
          arcarr = ARRAY [1..arcmax] OF INTEGER;
          del_type = ARRAY [0..T_hormax] OF INTEGER;
          wtd_delay_typ = ARRAY [0..T_hormax] OF REAL;
VAR     { mainly used for  simplex procedure }
          mm, mm1,nn1,ni,ii,jj,kk : INTEGER;
          Cm : row_matrix;
          Am : bigmatrix;
          inequalitym : inequalarray;
          bmi, bmo : matrix;
          ans : CHAR;
          period_cost : real;
          inp,out,out5 : TEXT;
        { used in mincost network }
          tt,T_, period_lenth,T_hor : 0..T_hormax;
          period_no : 1..10;
          no_nod, no_arc, nod,nod1,nod2 : nodetype;
          arc : arcaray;
          node : nodaray;
          degree: ARRAY [1..nodmax] OF INTEGER;
          ds : REAL;
          mincost_feasible : BOOLEAN;
        { Note that since all the arcs undirected i.e. two way
            indegree = outdegree = degree /2 }
            cumsum : INTEGER;
          point, fmovptr, rmovptr : ARRAY [1..nodmax] OF INTEGER;
          ftrace, rtrace : ARRAY [1..arcmax] OF INTEGER;
        { used for procedure find_assignment }
          dnn,snn, ino,jno, s_nod,cur_nod,d_nod : nodetype;
          dmn : ass_dem_nd;  spn : ass_sup_nd;
          slak, assign : sl_ass_typ;
          dem_nod_set : nodeset;
          arc_flo_set : arcset;
          cn,ft,arc_stor : INTEGER;
          ass1,ass2,ass, min_arc_capa,prev_min_arc_cap, flo_sent,
avail_tim,motion_tim,travl_tim,slack_tim,delay, rk : REAL;
          s_tim,d_tim, time : 0..T_hormax;
          ddeg,sdeg, spt,dpt : ARRAY [1..nodmax] OF INTEGER;
          arcs_aray : arcarr;
          { Empty wagon scheduling }
          no, istore,jstore,kstore,ptr : INTEGER;
          excl_arc : arcset;
          maxload : REAL; passed, tried : BOOLEAN;
          { output information }
          delay_freq : del_type;
          wtd_delay : wtd_delay_typ;
          tot_delay,total_cost : REAL;
          out1,out2,out3,out4 : TEXT;

FUNCTION smaller (a,b : REAL) : REAL;
BEGIN
          IF a < b
          THEN smaller := a
          ELSE smaller := b;
END;

PROCEDURE simplex
{------------------------------------------------------------}
(m,n:INTEGER; C:row_matrix; A:bigmatrix; inequality:inequalarray; b:matrix;
VAR b_o:matrix; VAR deci_vars:arcset; VAR feasibility : BOOLEAN);

LABEL 5,10,20;
TYPE  set_of_var = SET OF 1..nmax;
VAR   optimisation : optimisationtype;
      set_of_slack_var, set_of_artificial_var, set_of_altrn_optm_soln :
set_of_var;
```

```
        degeneracy_found : BOOLEAN;
         PROCEDURE  Resolve_degeneracy ;
        {---------  -----------------}
         VAR  column, i2 : INTEGER;
                tempratio, minratio : REAL;
         degeneracy_resolved : BOOLEAN;
         BEGIN { Resolve degeneracy }
                              degeneracy_resolved := FALSE; column := 0;
                WHILE  ( NOT degeneracy_resolved ) AND ( column <= m )  DO
                  BEGIN
                        column := column + 1;
                        minratio := 10000; { initialise }
                        FOR i2 := 1 TO stackptr DO
                        BEGIN
                             IF A_entering[indexro[i2]] > 0
                             THEN tempratio := B_inv[indexro[i2],column] /
A_entering[indexro[i2]]
                             ELSE IF A_entering[indexro[i2]] = 0
                                  THEN tempratio := 10000
                                  ELSE tempratio := -10000;
                             IF minratio > tempratio
                             THEN BEGIN
                                       minratio := tempratio;
                                       indexrow := indexro[i2];
                                  END;
                        END;
                        degeneracy_resolved := TRUE;
                  END { while }
         END; { Resolve degeneracy }

    BEGIN  { Find min_ratio }
        degeneracy_found := FALSE;
        min := -1;
        FOR i1:= 1 TO m DO
        IF ratio[i1] >= 0
        THEN IF (min < 0) OR (min > ratio[i1])
             THEN BEGIN
                       min := ratio[i1];
                       indexrow := i1;
                       degeneracy_found := FALSE;
```

```
                              stackptr := 1;
{ This step may be useful IF degeneracy is found at later stage }
                              indexro[stackptr] := i1;
                           END
                      ELSE IF min = ratio[i1]
                           THEN BEGIN
                                      degeneracy_found := TRUE;
                                      stackptr := stackptr + 1;
                                      indexro[stackptr] := i1;
                                 END;
            IF degeneracy_found = TRUE
            THEN Resolve_degeneracy ;
    END;    { Find min ratio }

PROCEDURE   Interchange_suffixes_of_variables(suffix1,suffix2 : INTEGER);
              VAR temp : INTEGER;
    BEGIN


              temp := non_bas_var[suffix1];
              IF suffix2 <= m
              THEN BEGIN
                         non_bas_var[suffix1] := basic_var[suffix2];
                         basic_var[suffix2] := temp
                   END
              ELSE BEGIN
                         non_bas_var[suffix1] := non_bas_var[suffix2];
                         non_bas_var[suffix2] := temp;
                   END;
      END;

PROCEDURE   Interchange(VAR x,y : REAL);
            VAR temp : REAL;
    BEGIN   temp := x;
            x := y;
            y := temp
    END;

PROCEDURE   Interchange_columns_of_variables
(entering_basis,leaving_basis : INTEGER);
VAR i1 : INTEGER;
```

```
          VAR temp : REAL;
     BEGIN
          FOR i1:= 1 TO m DO
          BEGIN
               temp := A[i1,entering_basis];
               A[i1,entering_basis] := A[i1,leaving_basis];
               A[i1,leaving_basis] := temp
          END
     END;


 BEGIN { Procedure simplex }
       REWRITE  (out,'outproje.dat');
       print_no := 1;
       m1 := m + 1;
       { To find out total no variables (nt) }
       nt := m+n;
       FOR i:= 1 TO m DO
       IF inequality[i] = '>'
       THEN nt := nt + 1;
       new_n := nt;
       optimisation := minimisation;

      { Columns indexed m+1 to new_n are for non_bas_var
                 and columns indexed 1 to m are basic_var }
      { Initialise non_bas_var and basic_var }
           FOR i:= 1 TO nt-m DO non_bas_var[m+i] := i;
           FOR i:= 1 TO m DO basic_var[i] := nt-m+i;
        CASE  optimisation OF
              maximisation : min_reducedcost :=-1 ;
              minimisation : max_reducedcost := 1;
        END;
              n1 := m + n;
              set_of_slack_var := []; set_of_artificial_var := [];
set_of_altrn_optm_soln := [];
       { Read A & b }
              count := 0;
              FOR i:=1 TO m DO
              BEGIN
                    IF (n1+1 <= nt)
                    THEN FOR j:= nl+1 TO nt DO
                         A[i,j] := 0;
                    CASE inequality[i] OF
                              '<' : BEGIN
                                             C[i] :=0;
                                             set_of_slack_var := set_of_slack_var
+ [basic_var[i]]
                                         END;
                              '>' : BEGIN
                                              count := count + 1 ;
                                              A[i,nl+count] := -1;
                                              C[nl+count] := 0;
                                              set_of_slack_var := set_of_slack_va
+ [non_bas_var[nl+count]];
                                            CASE optimisation OF
                                              maximisation : C[i] := {-M } -1000;
                                              minimisation : C[i] := { M }  1000
                                            END;
                                              set_of_artificial_var := set_of_art
+ [basic_var[i]]
                                         END;
                              '=' : BEGIN
                                            CASE optimisation OF
                                                 maximisation : C[i] := {-M }-100
                                                 minimisation : C[i] := { M } 100
                                            END;
                                            set_of_artificial_var :=
set_of_artificial_var + [basic_var[i]]
                                         END;
                    END; { case inequality[i] }
                    CB[i] := C[i];
             END;

    { Initialise coefficients of basic variables & B_inverse as unit matrix
             FOR i:= 1 TO m DO
             FOR j:= 1 TO m DO
             IF i=j
             THEN BEGIN
                       A[i,j] := 1;
                       B_inv[i,j]:= 1
```

```
                END
           ELSE BEGIN
                     A[i,j] := 0;
                     B_inv[i,j] := 0
                END;
        { Initially find the SIMPLEX Multipliers }
           FOR j := 1 TO m DO
           BEGIN
                PI[j] := 0;
                FOR i :=1 TO m DO
                PI[j] := PI[j] + CB[i] * B_inv[i,j]
           END;
      { Initially Find Reducedcosts for all variables }
            FOR j := 1 TO new_n DO
            BEGIN
                product := 0;
                FOR i:= 1 TO m DO
                product := product + PI[i] * A[i,j];
                reducedcost[j] := C[j] - product
            END;
        {  Initially take b_bar = b }
                FOR i := 1 TO m DO
                b_bar[i] := b[i];
               iteration_no := 0;  deleted := 0;
               CASE optimisation OF
                    maximisation : Find_max_reducedcost( max_reducedcost,
pivot_col );
                    minimisation : Find_min_reducedcost( min_reducedcost,
pivot_col );
               END;
5:   REPEAT
               iteration_no := iteration_no + 1 ;
        { Compute the updated coefficients of entering column }
               FOR i:= 1 TO m DO
               BEGIN
                     A_entering[i] := 0;
                     FOR j:= 1 TO m DO
                     A_entering[i] := A_entering[i] + B_inv[i,j]
* A[j,pivot_col]
               END;
        { Apply minimum ratio rule for determing leaving variable
              Find ratios }
                   FOR i:=1 TO m DO
                   BEGIN
                        IF abs(A_entering[i] - round(A_entering[i]))
<= 0.001
                        THEN A_entering[i] := round(A_entering[i]);
                        IF A_entering[i] > 0
                        THEN ratio[i] := b_bar[i] / A_entering[i]
                        ELSE IF A_entering[i] = 0
                             THEN ratio[i] := 10000
                             ELSE ratio[i] := -10000;
                   END;
              Find_min_ratio( min_ratio, pivot_row );
                IF min_ratio < 0
                THEN BEGIN
                          WRITELN (out,'This problem has
unbounded solution ');
                          WRITE ('Unbounded solution !!,
Program aborted ..');
                          GOTO 20
                     END;
        { Update B inverse matrix }
              FOR j:=1 TO m DO   {Update pivot row first}
                   IF A_entering[pivot_row] > 0
                   THEN B_inv[pivot_row,j] := B_inv[pivot_row,j] /
 A_entering[pivot_row];
              FOR i:=1 TO m DO   { Update all other rows }
              IF i <> pivot_row
              THEN FOR j:=1 TO m DO
                   B_inv[i,j] := B_inv[i,j] - B_inv[pivot_row,j]
 * A_entering[i];
         { Find b_bar }
              FOR i:=1 TO m DO
              BEGIN
                   b_bar[i] := 0;
                   FOR j:= 1 TO m DO
                   b_bar[i] := b_bar[i] + B_inv[i,j] * b[j]
              END;
        basic_col := pivot_row;
```

```
            {  Find SIMPLEX multipliers `PI's }
                   CB[basic_col] := C[pivot_col];
                   FOR j:= 1 TO m DO
                   BEGIN
                       PI[j] := 0;
                       FOR i :=1 TO m DO
                       PI[j] := PI[j] + CB[i] * B_inv[i,j]
                   END;
          {  Find Reducedcosts for all variables,
                 note that reducedcost will automatically be = 0
 for basic_var }
                 FOR j := 1 TO new_n DO
                 BEGIN
                     product := 0;
                     FOR i:= 1 TO m DO
                     product := product + PI[i] * A[i,j];
                     reducedcost[j] := C[j] - product
                 END;
             Interchange_suffixes_of_variables(pivot_col,basic_col);
             Interchange(C[pivot_col],C[basic_col]);
             Interchange_columns_of_variables(pivot_col,basic_col);
             Interchange(reducedcost[pivot_col],reducedcost[basic_col]);
             pc := non_bas_var[pivot_col];
         IF non_bas_var[pivot_col] IN set_of_artificial_var
         THEN BEGIN
{ Delete the artificial variable from subsequent considerations
once it has left the basis }
               old_n := nt - deleted;
               Interchange_suffixes_of_variables(pivot_col,old_n);
               Interchange(C[pivot_col],C[old_n]);
               Interchange_columns_of_variables(pivot_col,old_n);
               Interchange(reducedcost[pivot_col],reducedcost[old_n]);
               deleted := deleted + 1;
             END;
             feasibility := TRUE;
             FOR i:= 1 TO m DO IF b_bar[i] < 0
             THEN BEGIN
                     WRITELN (out,'Now problem is primal infeasible,
because one rhs constant b_bar is -ve');
                     CASE optimisation OF
```

```
                                 maximisation : FOR j:= ml TO new_n DO
                                                IF not(j IN set_of_slack_var)
AND not(j IN set_of_artificial_var) and
                                                (reducedcost[j] > 0)
                                                THEN BEGIN
                                                       WRITELN
(out,'It is also dual infeasible.  Hence can''t be solved further');
                                                       WRITE
('Infeasible solution !!, program aborted ..');
                                                       feasibility := FALSE;
                                                       GOTO 20;
                                                      END;
                                 minimisation : FOR j:= ml TO new_n DO
                                                IF not(j IN set_of_slack_var)
AND not(j IN set_of_artificial_var) and
                                                (reducedcost[j] < 0)
                                                THEN  BEGIN
                                                        WRITELN
(out,'It is also dual infeasible.  Hence can''t be solved further');
                                                        WRITE
('Infeasibible solution !!, program aborted ..');
                                                        feasibility := FALSE;
                                                        GOTO 20
                                                      END;
                           END;
                           WRITELN (out,'However you may solve it further,
by dual simlex method');
                           WRITE ('Primal infeasible !!, program aborted ..');
                           GOTO 20;
                      END;
                 new_n := nt - deleted;
                 CASE optimisation OF
                      maximisation : BEGIN
                                           Find_max_reducedcost
(max_reducedcost,pivot_col);
                                           IF abs( max_reducedcost - round
(max_reducedcost)) <= 0.001
                                           THEN max_reducedcost :=
round(max_reducedcost);
                                           IF max_reducedcost = 0
```

```
                                        THEN set_of_altrn_optm_soln :=
set_of_altrn_optm_soln + [pc] {[non_bas_var[pivot_col]]};
                                   END;
                    minimisation : BEGIN
                                        Find_min_reducedcost
(min_reducedcost,pivot_col);
                                        IF abs( min_reducedcost -
round(min_reducedcost)) <= 0.001
                                        THEN min_reducedcost :=
round(min_reducedcost);
                                        IF min_reducedcost = 0
                                        THEN set_of_altrn_optm_soln :
= set_of_altrn_optm_soln + [pc] {[non_bas_var[pivot_col]]};
                                   END
                 END;
        UNTIL ( max_reducedcost <= 0 ) OR ( min_reducedcost >= 0 );
              { Further minimisation of objective function is not possible
i.e. when optimum solution is obtained };
        IF set_of_altrn_optm_soln <> []
        THEN BEGIN
                   WRITE (out,'Set of alternate optimum soln contains : ');
                   FOR i:= 1 TO nt DO
                   IF i IN set_of_altrn_optm_soln
                   THEN WRITE (out,'X',i:2);  WRITELN (out);
              END;
10:    { Compute the value of minimum value of objective function}
                   Z := 0;
              FOR i:= 1 TO m DO
              Z := Z + CB[i] * b_bar[i];
       { Print output }
       CASE optimisation OF
          maximisation : WRITELN(out,'Maximum objective function value
is Z = ',Z:6:2);
          minimisation : WRITELN(out,'Minimum objective function value
is Z = ',Z:6:2)
       END;
         FOR i := 1 TO m DO
         b_o[i] := 0; { initialise before returning output }
            WRITELN(out,'Optimum solution is ');
            FOR i := 1 TO m DO
          BEGIN
                 IF NOT ((basic_var[i] IN set_of_slack_var) OR
(basic_var[i] IN set_of_artificial_var)) AND (b_bar[i] > 0.001)
                 THEN BEGIN
                         WRITE(out,'  X',basic_var[i]:2,' = ',b_bar[i]:4:2);
                         deci_vars := deci_vars + [basic_var[i]];
                         b_o[ basic_var[i]] := b_bar[i];
                      END;
             END;  WRITELN(out);
             FOR i := 1 TO m DO
             IF (basic_var[i] IN set_of_artificial_var) AND (b_bar[i] <> 0 )
             THEN BEGIN
                       WRITELN (out,'Since artificial variable is present
in the optimum solution ');
                       WRITELN (out,'PROBLEM IS INFISIBLE');
                       WRITELN(out,'  X',basic_var[i]:2,' = ',b_bar[i]:4:2);
                       WRITELN ('Infisibility !!, program aborted ..');
                       feasibility := FALSE;
                       GOTO 20;
                  END
             ELSE feasibility := TRUE;
             CASE optimisation OF
                   maximisation : IF (max_reducedcost = 0)
AND not(non_bas_var[pivot_col] IN set_of_altrn_optm_soln)
                                  THEN BEGIN WRITELN (out,
'Finding out another alternate optimum soln ');GOTO 5 END;
                   minimisation : IF (min_reducedcost = 0)
AND not(non_bas_var[pivot_col] IN set_of_altrn_optm_soln)
                                  THEN BEGIN WRITELN (out,'Finding out
another alternate optimum soln ');GOTO 5 END;
             END;
             WRITE ('Complete solution has been written in the
file outproje.dat');
20:          READLN;  {CLOSE(out);}
END;  { Procedure  Simplex }

FUNCTION shortest_dist ( s, t : nodetype): REAL;
{-----------------------------------------------------------------}
VAR     maxdist, temp_dist : REAL;
        Nset, temp, perm : nodeset;
        shordist : ARRAY [1..nodmax] OF REAL;
        i,j, arc_no : INTEGER;
        store, storl : nodetype;

FUNCTION  min_temp_dist_nod : nodetype ;
VAR minimum : REAL;  k : nodetype;
BEGIN
        minimum := maxdist;
        FOR k := 1 TO no_nod DO
        IF NOT (k IN perm)
        THEN  IF shordist[k] < minimum
              THEN BEGIN
                        minimum := shordist[k];
                        min_temp_dist_nod := k;
                   END;
END; { min_temp_dist_nod }

BEGIN { function shortest dist }
        Nset := [1..no_nod];
        perm := [s];  temp := Nset - [s];
        IF NOT ((s IN Nset) AND (t IN Nset))
        THEN BEGIN
                WRITELN ('Node out of network data range !!');
                GOTO 300;
             END;
        { Set shortest dist for other nodes = sum of all dist.s in network }
                maxdist := 0;
                FOR  i:= 1 TO 2*no_arc DO
                        maxdist := maxdist + arc[i].dist;
                FOR i  := 1 TO no_nod DO
                        shordist[i] := maxdist;
                shordist[s] := 0;
        store := s;
        WHILE NOT (t IN perm) DO
        BEGIN
                FOR j:= point[store] TO point[store+1] -1 DO
                BEGIN { Get the node in the adjuscency list of node 'store'
                        called 'storl', update its shordist if required }
                        arc_no := ftrace[j];
                        temp_dist := shordist[store] + arc[arc_no].dist;
```

```
                              storl  := arc[arc_no].node2;
                              IF storl IN temp
                              THEN shordist[storl] := smaller (shordist[storl],
temp_dist);
                    END; { for adjuscent nodes update shordist }
                    store := min_temp_dist_nod;
                    temp := temp - [store];
                    perm := perm + [store];
          END; { while not (t in perm) DO }
          shortest_dist := shordist[t];
END; { function  Shortest dist }

PROCEDURE  Find_Assignment (     nodn,nosn : nodetype;
{----------------------------}   dn : ass_dem_nd; sn : ass_sup_nd;
                                 slack : sl_ass_typ;
                             VAR assignment : sl_ass_typ);
VAR      ip, jp, ro,co, count : INTEGER;
         assigned :  ARRAY [1..nodmax,1..nodmax] OF BOOLEAN;
         totslack, small : REAL;

PROCEDURE Max_slack (VAR row, col: INTEGER);
VAR      i,j  : INTEGER;  max : REAL;
PROCEDURE  braketie;
VAR      penalty1, penalty2 : REAL;
FUNCTION  next_max_in_row (iro : INTEGER) : REAL;
VAR      k  : INTEGER;  nextmax : REAL;
BEGIN
         nextmax := 0;
         FOR k := 1 TO nodn DO
         IF (nextmax < slack[iro,k]) AND (slack[iro,k] < max)
AND NOT assigned[iro,k]
         THEN nextmax := slack[iro,k];
         next_max_in_row := nextmax;
END;
FUNCTION  next_max_in_col (ico : INTEGER) : REAL;
VAR      k  : INTEGER;  nextmax : REAL;
BEGIN
         nextmax := 0;
         FOR k := 1 TO nosn DO
         IF (nextmax < slack[k,ico]) AND (slack[k,ico] < max)
```

```
AND NOT assigned[k,ico]
        THEN nextmax := slack[k,ico];
        next_max_in_col := nextmax;
END;
BEGIN { braketie }
        IF j = col
        THEN BEGIN
                penalty1 := max - next_max_in_row(i);
                penalty2 := max - next_max_in_row(row);
                IF penalty1 > penalty2
                THEN row := i;
             END;
        IF i = row
        THEN BEGIN
                penalty1 := max - next_max_in_col(j);
                penalty2 := max - next_max_in_col(col);
                IF penalty1 > penalty2
                THEN col := j;
             END;
END;  { braketie }
BEGIN { procedure max_slack }
        max := 0;  row := 1; col := 1;
        FOR i := 1 TO nosn DO
        FOR j := 1 TO nodn DO
        IF (max < slack[i,j]) AND (dn[j].dem> 0.001) AND (sn[i].sup > 0.001)
        THEN BEGIN
                max := slack[i,j];
                row := i;
                col := j;
             END
        ELSE IF (round(max) = round(slack[i,j])) AND
                 (dn[j].dem > 0.001) AND (sn[i].sup > 0.001)
             THEN IF NOT assigned[i,j]
                  THEN  IF (i=row) OR (j=col)
                        THEN braketie
                        ELSE BEGIN
                                row := i;
                                col := j;
                             END;
END; { procedure max_slack }
```

```
FUNCTION  demand_satisfied : BOOLEAN;
VAR       i : INTEGER;
BEGIN
          demand_satisfied := TRUE;
          FOR i := 1 TO nodn DO
          IF dn[i].dem  > 0.001
          THEN demand_satisfied := FALSE;
END;

PROCEDURE  print_assignment;
VAR       i,j : INTEGER;
BEGIN
          FOR i := 1 TO nosn DO
          BEGIN
                    FOR j := 1 TO nodn DO
                    WRITE (assignment[i,j]:4:0);
                    WRITELN;
          END;
          READLN;
END;

BEGIN { Procedure Find_assignment }
          FOR ip := 1 TO nosn DO
          FOR jp := 1 TO nodn DO
          BEGIN
                    assignment[ip,jp] := 0;
                    assigned[ip,jp] := FALSE;
          END;
          count := 0;
          REPEAT
                    count := count + 1;
                    Max_slack(ro,co);
                    small := smaller(sn[ro].sup,dn[co].dem );
                    sn[ro].sup := sn[ro].sup - small;
                    dn[co].dem  := dn[co].dem  - small;
                    Assignment[ro,co] := small;
                    assigned[ro,co] := TRUE;
                    { print_assignment;}
          UNTIL  (demand_satisfied) OR (count >= 200);
```

```
        IF count = 200 THEN WRITELN ('Demand can''t be satisfied **');
        totslack := 0;
        FOR ip := 1 TO nosn DO
        FOR jp := 1 TO nodn DO
                totslack := totslack + slack[ip,jp]*assignment[ip,jp];
        WRITELN ('Total slackness : ',totslack:5:1);
END; { Procedure Find_assignment }


PROCEDURE  Find_short_assign (    nodn,nosn : nodetype;
{---------------------------}    dn : ass_dem_nd; sn : ass_sup_nd;
                                 short : sl_ass_typ;
                              VAR assignment : sl_ass_typ);
VAR     ip, jp, ro,co, count : INTEGER;
        assigned :  ARRAY [1..nodmax,1..nodmax] OF BOOLEAN;
        totshort, small : REAL;

PROCEDURE Min_short (VAR row, col: INTEGER);
VAR     i,j : INTEGER;  min : REAL;
PROCEDURE  braketie;
VAR     penalty1, penalty2 : REAL;
FUNCTION  next_min_in_row (iro : INTEGER) : REAL;
VAR     k : INTEGER;  nextmin : REAL;
BEGIN
        nextmin := 1000;
        FOR k := 1 TO nodn DO
        IF (nextmin > short[iro,k]) AND (short[iro,k] > min) AND
NOT assigned[iro,k]
        THEN nextmin := short[iro,k];
        next_min_in_row := nextmin;
END;
FUNCTION  next_min_in_col (ico : INTEGER) : REAL;
VAR     k : INTEGER;  nextmin : REAL;
BEGIN
        nextmin := 1000;
        FOR k := 1 TO nosn DO
        IF (nextmin > short[k,ico]) AND (short[k,ico] > min)
AND NOT assigned[k,ico]




        THEN nextmin := short[k,ico];
        next_min_in_col := nextmin;
```

```
   END;
   BEGIN { braketie }
           IF j = col
           THEN BEGIN
                     penalty1 := next_min_in_row(i) - min;
                     penalty2 := next_min_in_row(row) - min;
                     IF penalty1 > penalty2
                     THEN row := i;
                 END;
          IF i = row
          THEN BEGIN
                     penalty1 := next_min_in_col(j) - min;
                     penalty2 := next_min_in_col(col) - min;
                     IF penalty1 > penalty2
                     THEN col := j;
                 END;
  END;  { braketie }
  BEGIN { procedure Min_short }
          min := 1000;  row := 1; col := 1;
          FOR i := 1 TO nosn DO
          FOR j := 1 TO nodn DO
          IF (min > short[i,j]) AND (dn[j].dem> 0.001)
  AND (sn[i].sup > 0.001) { preferably check time feasibility }
          THEN BEGIN
                    min := short[i,j];
                    row := i;
                    col := j;
                END
         ELSE IF (round(min) = round(short[i,j])) AND
                    (dn[j].dem > 0.001) AND (sn[i].sup > 0.001)
                THEN IF NOT assigned[i,j]
                     THEN  IF (i=row) OR (j=col)
                           THEN braketie
                           ELSE BEGIN
                                   row := i;
                                   col := j;
                                END;
 END; { procedure min_short }



 FUNCTION  demand_satisfied : BOOLEAN;
```

```
VAR      i : INTEGER;
BEGIN
         demand_satisfied := TRUE;
         FOR i := 1 TO nodn DO
         IF dn[i].dem  > 0.001
         THEN demand_satisfied := FALSE;
END;

PROCEDURE  print_assignment;
VAR      i,j  : INTEGER;
BEGIN
         FOR i := 1 TO nosn DO
         BEGIN
                  FOR j := 1 TO nodn DO
                  WRITE (assignment[i,j]:4:0);
                  WRITELN;
         END;
         READLN;
END;

BEGIN { Procedure Find_short_assign }
         FOR ip := 1 TO nosn DO
         FOR jp := 1 TO nodn DO
         BEGIN
                  assignment[ip,jp] := 0;
                  assigned[ip,jp] := FALSE;
         END;
         count := 0;
         REPEAT
                  count := count + 1;
                  Min_short(ro,co);
                  small := smaller(sn[ro].sup,dn[co].dem );
                  sn[ro].sup := sn[ro].sup - small;
                  dn[co].dem  := dn[co].dem  - small;
                  Assignment[ro,co] := small;
                  assigned[ro,co] := TRUE;
                  { print_assignment;}
         UNTIL  (demand_satisfied) OR (count >= 200);
         IF count = 200 THEN WRITELN ('Demand can''t be satisfied **');



         totshort := 0;
```

```
        FOR ip := 1 TO nosn DO
        FOR jp := 1 TO nodn DO
              totshort := totshort + short[ip,jp]*assignment[ip,jp];
        WRITELN ('Total shortness : ',totshort:5:1);
END;  { Find_short_assign }

PROCEDURE find_short_ex_aray ( s, t : nodetype; ex_arc : arcset;
                VAR short_aray : arcarr; VAR stackptr : INTEGER );
{-------------------------------------------------------------}
LABEL   20;
VAR     maxdist, temp_dist : REAL;
        Nset, temp, perm : nodeset;
        shordist : ARRAY [1..nodmax] OF REAL;
        i,j, arc_no : INTEGER;
        store, storl : nodetype;

FUNCTION  min_temp_dist_nod : nodetype ;
VAR minimum : REAL;  k : nodetype;
BEGIN
        minimum := maxdist;
        FOR k := 1 TO no_nod DO
        IF NOT (k IN perm)
        THEN  IF shordist[k] < minimum
              THEN BEGIN
                      minimum := shordist[k];
                      min_temp_dist_nod := k;
                   END;
END; { min_temp_dist_nod }

BEGIN { Procedure find short aray }
        Nset := [1..no_nod];
        perm := [s];  temp := Nset - [s];
        IF NOT ((s IN Nset) AND (t IN Nset))
        THEN BEGIN
              WRITELN ('Node out of network data range !!');
              GOTO 300;
           END;
        { Set shortest dist for other nodes = sum of all dist.s in network
              maxdist := 0;




              FOR  i:= 1 TO 2*no_arc DO
```

```
                    maxdist := maxdist + arc[i].dist;
               FOR i := 1 TO no_nod DO
                    shordist[i] := maxdist;
               shordist[s] := 0;
        store := s;
        WHILE NOT (t IN perm) DO
        BEGIN
               FOR j:= point[store] TO point[store+1] -1 DO
               BEGIN { Get the node in the adjuscency list of node 'store'
                    called 'storl', update its shordist if required }
                    arc_no := ftrace[j];
                    IF not(arc_no IN ex_arc)
                    THEN temp_dist := shordist[store] + arc[arc_no].dis
                    ELSE temp_dist := 10000;
                    storl := arc[arc_no].node2;
                    IF storl IN temp
                    THEN shordist[storl] := smaller (shordist[storl],
  temp_dist);
               END; { For adjuscent nodes update shordist }
               store := min_temp_dist_nod;
               temp := temp - [store];
               perm := perm + [store];
        END; { while not (t in perm ) DO }
        { find the arcs in the shortest path }
        perm := [t];  temp := Nset - [t];
        store := t; stackptr := 0;
        WHILE NOT (s IN perm) DO
        BEGIN
               FOR j:= point[store] TO point[store+1] -1 DO
               BEGIN
                    arc_no := rtrace[j];
                    IF (shordist[ arc[arc_no].nodel] = shordist[store]
  - arc[arc_no].dist) AND NOT (arc_no IN ex_arc)
                    THEN BEGIN
                         store := arc[arc_no].nodel;
                         GOTO 20;
                       END;
               END;



20:            stackptr := stackptr + 1;
               short_aray[stackptr] := arc_no;
```

```
                temp := temp - [store];
                perm := perm + [store];
        END; { while not (s in perm ) do }

END; { Procedure find short ex aray }

PROCEDURE Update_delay_info(dele : REAL; flo : REAL;
{---------------------------}
                VAR del_aray : del_type;  VAR wt_dele : wtd_delay_typ;
                VAR stackptr : INTEGER);
VAR     int_del : INTEGER;
BEGIN
        int_del := round(dele);
        IF int_del > stackptr
        THEN stackptr := int_del;
        del_aray[int_del] := del_aray[int_del] + round(flo);
        wt_dele[int_del] := wt_dele[int_del] + dele*flo;
END;


BEGIN { main }
        WRITE ('Do you want to begin with infinite capacity of arcs? ');
        READLN (ans);
        IF (ans = 'y') OR (ans = 'Y')
        THEN RESET (inp,'inpsup.dat')
        ELSE    RESET (inp,'inpsupcap.d');
        READLN (inp,no_nod, no_arc);
        FOR ii := 1 TO no_arc DO
        WITH arc[ii] DO
        IF (ans = 'y') OR (ans = 'Y')
        THEN READLN (inp, node1, node2,dist)
        ELSE    READLN (inp, node1, node2,dist,capa);
        FOR ii := 1 TO no_arc DO
        WITH arc[ii] DO
        BEGIN
                cost_g := dist / 10;
                cost_w := cost_g /1.5;
                dist := dist / (50*4);
                IF (ans = 'y') OR (ans = 'Y')




                THEN capa := 1000;
        END;
        FOR ii := no_arc+1 TO 2*no_arc DO
        BEGIN
                arc[ii].node1 := arc[ii-no_arc].node2;
                arc[ii].node2 := arc[ii-no_arc].node1;
                arc[ii].cost_g := arc[ii-no_arc].cost_g;
                arc[ii].cost_w := arc[ii-no_arc].cost_w;
                arc[ii].dist := arc[ii-no_arc].dist;
                arc[ii].capa := arc[ii-no_arc].capa;
        END;
25:     WRITE ('Give period length : '); READLN (period_lenth);
        WRITE ('Give time horrizon : '); READLN (T_hor);
        IF T_hor < period_lenth
        THEN BEGIN
                WRITELN ('Time horrizon must be > period lenth ');
                GOTO  25;
             END;
        FOR ii := 1 TO 2*no_arc DO
        WITH arc[ii] DO { initialise }
        FOR tt := 1 TO T_hormax DO
        BEGIN
                res_cap[tt] := capa;
                load[tt] := 0;
        END;
        FOR ii := 1 TO no_nod DO
        WITH node[ii] DO { Initialise }
                FOR tt := 1 TO T_hor DO
                BEGIN
                        dema_supl[tt] := 0;
                        requ_avai[tt] := 0;
                END;

        WHILE NOT EOF (inp) DO
        BEGIN
                READ (inp, nod,ds,time);
                WITH node[nod] DO




                dema_supl[time*6] := ds;
                IF EOLN (inp)
                THEN READLN(inp);
        END;
```

```
RESET (inp,'inpdem.dat');
WHILE NOT EOF (inp) DO
BEGIN
          READ (inp, nod,ds,time);
          WITH node[nod] DO
          dema_supl[time*6] := -ds;
          IF EOLN (inp)
          THEN READLN(inp);
END;
RESET (inp,'inprowag.dat'); { Check and delet to read it once }
WHILE NOT EOF (inp) DO
BEGIN
          READLN (inp,nod,ds,time);
          WITH node[nod] DO
          requ_avai[time*6] := requ_avai[time*6] + ds;
END;
CLOSE (inp);

FOR ii := 1 TO no_nod DO
WITH node[ii] DO
          cum_dem_sup_g := 0;
FOR ii := 1 TO no_nod DO
          degree[ii] := 0;
FOR ii := 1 TO 2*no_arc DO
WITH arc[ii] DO
          degree[node2] := degree[node2] + 1;
cumsum := 1;
FOR ii := 1 TO no_nod+1 DO
BEGIN
          point[ii] := cumsum;
          cumsum := cumsum + degree[ii];
END;
FOR ii := 1 TO no_nod+1 DO {Initialisation }
BEGIN
          fmovptr[ii] := point[ii];
          rmovptr[ii] := point[ii];




END;
FOR ii := 1 TO 2*no_arc DO
WITH arc[ii] DO
BEGIN
```

```
            ftrace[ fmovptr[node1]] := ii;
            fmovptr[node1] := fmovptr[node1] + 1;
            rtrace[ rmovptr[node2]] := ii;
            rmovptr[node2] := rmovptr[node2] + 1;
     END;
     FOR ii := 0 TO T_hor DO
     BEGIN
            delay_freq[ii] := 0;
            wtd_delay[ii] := 0;
     END;
     REWRITE (out2,'outroute.dat');
     REWRITE (out5,'outcost.dat');
     WRITELN (out5,'Period Goods Cost  Empty Wagon Cost');
     period_no := 1;
50:  WRITELN; WRITELN ('Goods wagon scheduling');
     WRITELN ('Period no : ',period_no:2);
     T_ := period_no* period_lenth;

     FOR ii := 1 TO no_nod DO
     WITH node[ii] DO
     BEGIN
            FOR tt := (period_no-1)*period_lenth + 1
                           to period_no*period_lenth DO
            cum_dem_sup_g := cum_dem_sup_g + dema_supl[tt];
            IF cum_dem_sup_g < -0.001
            THEN dem_nod_set := dem_nod_set + [ii]
     END;
     mm := no_nod + no_arc * 2; { No. of constraints }
     ni := 2*no_arc; { No of decision variables }
     FOR ii := 1 TO 2*no_arc DO
            cm[mm+ii] := arc[ii].cost_g;
     FOR ii := 1 TO mm DO
     FOR jj := mm+1 TO mm+ni DO
            Am[ii,jj] := 0;
     FOR ii := 1 TO no_nod DO
     WITH node[ii] DO
     BEGIN
            { For all the arcs incoming at node ii  }
            FOR jj := point[ii] TO point[ii+1] -1 DO




            BEGIN
```

```
                    kk  := mm + rtrace[jj];
                    IF cum_dem_sup_g <= 0
                    THEN Am[ii,kk] := 1
                    ELSE Am[ii,kk] := -1;
          END;
          { For all the outgoing arcs from node ii }
          FOR jj := point[ii] TO point[ii+1] -1 DO
          BEGIN
                    kk  := mm + ftrace[jj];
                    IF cum_dem_sup_g <= 0
                    THEN Am[ii,kk] := -1
                    ELSE Am[ii,kk] := 1;
          END;
          IF cum_dem_sup_g < -0.001
          THEN inequalitym[ii] := '>'
          ELSE      IF abs(cum_dem_sup_g) <= 0.001
                    THEN inequalitym[ii] := '='
                    ELSE inequalitym[ii] := '<';
          bmi[ii] := abs(node[ii].cum_dem_sup_g);
END; { for ii := 1 TO no_nod do }
WRITELN ('Node cum_dem_sup_g');
FOR ii := 1 TO no_nod DO
WRITELN (ii:4,node[ii].cum_dem_sup_g:8:1);
jj := 0;
FOR ii := 1 TO no_arc*2 DO
BEGIN
          jj := jj + 1;
          Am[ii+no_nod,mm+jj] := 1;
          inequalitym[ii+no_nod] := '<';
          bmi[no_nod+ii] := arc[ii].capa;
END;
simplex (mm,ni,Cm,Am,inequalitym,bmi, bmo,arc_flo_set,mincost_feasible);
period_cost := 0;
IF mincost_feasible THEN
          FOR ii := 1 TO 2*no_arc DO
          period_cost := period_cost + bmo[ii]*arc[ii].cost_g;
snn := 0; { No.of sup nodes }
dnn := 0; { No.of dem nodes }




FOR ii := 1 TO no_nod+1 DO
```

```
            ddeg[ii] := 0;
            sdeg[ii] := 0;
    END;
    FOR ii := 1 TO no_nod DO
    WITH node[ii] DO
    BEGIN
            FOR tt := 1 TO T_ DO
            IF dema_supl[tt] < -0.001
            THEN BEGIN
                    dnn := dnn + 1;
                    ddeg[ii] := ddeg[ii] + 1;
                    WITH dmn[dnn] DO
                    BEGIN   nd := ii;
                            dem := abs(dema_supl[tt]);
                            tim := tt;
                    END;
                 END
            ELSE IF dema_supl[tt] > 0.001
                 THEN BEGIN
                            snn := snn + 1;
                            sdeg[ii] := sdeg[ii] + 1;
                            WITH spn[snn] DO
                            BEGIN   nd := ii;
                                    sup := dema_supl[tt];
                                    tim := tt;
                            END;
                      END;
    END;
    IF mincost_feasible
    THEN BEGIN
            FOR ii := 1 TO snn DO
            FOR jj := 1 TO dnn DO
            BEGIN
                    ino := spn[ii].nd;
                    jno := dmn[jj].nd;
                    slak[ii,jj] := dmn[jj].tim - spn[ii].tim
                                   - shortest_dist(ino,jno);

            END;
            find_assignment(dnn,snn,dmn,spn,slak,assign);



        END
```

```
ELSE BEGIN
          FOR ii  := 1 TO snn DO
          FOR jj  := 1 TO dnn DO
          BEGIN
                    ino  := spn[ii].nd;
                    jno  := dmn[jj].nd;
                    slak[ii,jj] := shortest_dist(ino,jno);
          END;
          find_short_assign(dnn,snn,dmn,spn,slak,assign);
          FOR ii  := 1 TO 2*no_arc DO
          WITH arc[ii] DO
          FOR tt  := (period_no-1)*period_lenth+1 TO
                           period_no*period_lenth DO
                   res_cap[tt] := capa/6;
      END;
FOR ii  := 1 TO 2*no_arc DO
WITH arc[ii] DO
BEGIN
          tmp_capa  := bmo[ii];
          IF tmp_capa < 0.001
          THEN deleted := TRUE
          ELSE deleted := FALSE;
END;


cumsum  := 1;
FOR ii  := 1 TO no_nod+1 DO
BEGIN
          dpt[ii] := cumsum;
          cumsum := cumsum + ddeg[ii];
END;
cumsum  := 1;
FOR ii  := 1 TO no_nod +1 DO
BEGIN
          spt[ii] := cumsum;
          cumsum := cumsum + sdeg[ii];
END;
FOR ii  := 1 TO dnn DO
          WRITE (dmn[ii].nd:3);
```

```
                 WRITELN;
        WRITE ('           ');
        FOR ii  := 1 TO dnn DO
                 WRITE (dmn[ii].dem:3:0);
                 WRITELN;
        WRITE ('           ');
        FOR ii  := 1 TO dnn DO
                 WRITE (dmn[ii].tim:3);
                 WRITELN;
        FOR s_nod  := 1 TO no_nod DO
        FOR jj:= spt[s_nod] TO spt[s_nod+1]-1 DO
        BEGIN
                 WRITE (spn[jj].nd:3, spn[jj].sup:3:0,spn[jj].tim:3);
                 FOR cur_nod  := 1 TO no_nod DO
                 FOR kk:= dpt[cur_nod] TO dpt[cur_nod+1]-1 DO
                 WRITE (assign[jj,kk]:3:0);
                 WRITELN;
        END;

        WRITELN (out2,'Goods    Period no :  ',period_no:2);
        WRITELN (out2,'Flow      ----     Node     Time     ----');
        cn  := 1; s_nod  := 1;
        IF mincost_feasible THEN
        WHILE (dem_nod_set <> []) AND (cn<400) DO
        BEGIN
                 WHILE (node[s_nod].cum_dem_sup_g < 0.001)
AND (s_nod <= no_nod) DO
                          s_nod  := s_nod + 1;
                 cur_nod  := s_nod;
                 WRITELN ('cn : ',cn:2,'   s_nod', s_nod:4,'   sup
',node[s_nod].cum_dem_sup_g:4:0);
                 min_arc_capa  := 1000;
                 no  := 0;
                 travl_tim  := 0; flo_sent  := 0;
                 FOR ii  := 1 TO 2*no_arc DO
                          arc[ii].enter  := TRUE;
75:              REPEAT
                 WRITELN ('cur_nod :  ',cur_nod:4);



                 WRITE ('ft :');
                 FOR ii  := point[cur_nod] TO point[cur_nod+1]-1 DO
```

```
                WITH arc[ftrace[ii]] DO
                BEGIN
                        ft := ftrace[ii]; WRITE (ft:4);
                        IF (ft IN arc_flo_set) AND NOT deleted AND enter
                        THEN BEGIN
                                WRITELN ('  enters');
                                arc_stor := ft;
                                no := no + 1;
                                arcs_aray[no] := ft;
                                travl_tim := travl_tim + dist;
                                prev_min_arc_cap := min_arc_capa;
                                min_arc_capa := smaller(arc[ft].tmp_capa,
min_arc_capa);
                                IF node2 IN dem_nod_set
                                THEN BEGIN
                                        ass := 0;
                                FOR jj:= spt[s_nod] TO spt[s_nod+1]-1 DO
                                FOR kk:= dpt[node2] TO dpt[node2+1]-1 DO
                                        IF assign[jj,kk] > 0.001
                                        THEN BEGIN
                                                ass := assign[jj,kk];
                                                jstore := jj;
                                                kstore := kk;
                                                GOTO 200;{exit for loop}
                                             END;
200:                                    WRITELN ('Demand nod',node2:4);
                                        IF ass > 0.001
                                        THEN BEGIN
                                                avail_tim := dmn[kstore].
tim-spn[jstore].tim;
                                                IF avail_tim < travl_tim
                                                THEN BEGIN
                                                        WRITE ('time
infeasibility  ');
                                                        delay := travl_tim
- avail_tim;
                                                        WRITELN ('delay ',
delay:5:2);
                                                        s_tim := spn[jstore
tim;
```

```
                                                            d_tim := s_tim +
round(travl_tim);
                                                         END
                                                    ELSE BEGIN
                                      d_tim := dmn[kstore].tim;
                                                            delay := 0;
                                      IF d_tim - round(travl_tim)
> 0.001
                                                            THEN s_tim := d_tim
-round(travl_tim)
                                                            ELSE s_tim := 0;
                                                         END;
                                         flo_sent := smaller(min_arc_capa, ass);
                                         Update_delay_info(delay, flo_sent,
 delay_freq,wtd_delay,ptr);
                                   assign[jstore,kstore] := assign[jstore,kstore]
- flo_sent;
                                                   WITH dmn[kstore] DO
                                                   BEGIN
                                                         dem := dem - flo_sent;
                                                         FOR jj := 1 TO snn DO
                                                           assign[jj,kstore] :=
assign[jj,kstore] - flo_sent;
                                                    node[nd].dema_supl[tim] :=
node[nd].dema_supl[tim] + flo_sent;
                                                  node[nd].requ_avai[d_tim] :=
node[nd].requ_avai[d_tim] + flo_sent;
                                                          WRITELN ('tim d ',
d_tim:2;
                                                   END;
                                                   WITH spn[jstore] DO
                                                   BEGIN
                                                           sup := sup - flo_sent
                                                           FOR kk := 1 TO dnn DO
                                                           assign[jstore,kk] :=
assign[jstore,kk] - flo_sent;
                                                      WRITELN ('tim s ',s_tim:2);
                                                 node[nd].dema_supl[tim] :=
node[nd].dema_supl[tim] - flo_sent;
```

```
                                    THEN node[nd].requ_avai[s_tim] :=
ode[nd].requ_avai[s_tim] - flo_sent;

                                           END;
                                           {WRITE arcs aray ok fwd}
                                           kk := d_tim;
                                           rk := kk;
                                           FOR jj := no DOWNTO 1 DO
                                           WITH arc[ arcs_aray[jj]]DO
                                           BEGIN
                                          FOR tt := kk DOWNTO
kk-round(dist) DO

                                                 BEGIN
                                                        res_cap[tt] :=
res_cap[tt]-flo_sent;

                                                        load[tt] :=
load[tt] + flo_sent;

                                                 END;
                                                 WRITELN ('Arc no ',
arcs_aray[jj]:2);

                                                 WRITELN ('Res capa ',
res_cap[tt]:3:0,'  Load ',load[tt]:3:0);

                                                 WRITE ('rk : ',rk:3:0);
                                                 rk := rk - dist;
                                                 WRITELN(rk:3:0);
                                                 kk := round(rk);
                                        END;
                                        WRITELN('flo sent ',flo_sent:4:0);
                                        END { ass > 0.001 }
                                    ELSE GOTO 250;{ ass <= 0.001 }
                                 END
                             ELSE GOTO 250;{node2 not in dem_nod_set}
                             node[s_nod].cum_dem_sup_g :=
node[s_nod].cum_dem_sup_g - flo_sent;
                             FOR jj := 1 TO no DO
                             WITH arc[arcs_aray[jj]] DO
                                 BEGIN
                                    tmp_capa := tmp_capa - flo_sent;
                                    IF tmp_capa < 0.001
                                    THEN deleted := TRUE;
                                    node[node2].cum_dem_sup_g :=
node[node2].cum_dem_sup_g + flo_sent;

                                    IF node[node2].cum_dem_sup_g > -0.001
                                    THEN dem_nod_set := dem_nod_set - [node2];
                                    WRITELN ('flag 2 ');
                                    GOTO 100;
                            END; (IF ft in arc_flo_st & not arc[ft].deletd)
                   END;  (for ii := fpoint.. & WITH arc[ii] do)
                   WRITELN;
                   (s_nod had surplus suplies which could not be absorbed
WITH current assignments)
                   IF s_nod <= no_nod
                   THEN IF no > 0
                        THEN BEGIN
                                    arc[arcs_aray[no]].enter := FALSE;
                                    min_arc_capa := prev_min_arc_cap;
                                    cur_nod := arc[arcs_aray[no]].nodel;
                                    no := no - 1;
                                    WRITELN ('Backtracking');
                                    GOTO 75;
                              END
                        ELSE  BEGIN
                                    s_nod := s_nod + 1;
                                WRITELN ('Futile search,
s_nod Quit.. surplus assumed')
                                END
                   ELSE BEGIN ( s_nod > no_nod )
                           WRITELN ('Failure with current assignments,
discarding them');
                           s_nod := 1;
                           FOR ii := 1 TO snn DO
                           FOR jj := 1 TO dnn DO
                           IF (spn[ii].sup>0.001) AND (dmn[jj].dem>0.001)
                           THEN assign[ii,jj] := smaller
(spn[ii].sup,dmn[jj].dem);
                        END;
                   cn := cn + 1;
```

```
              cn := cn + 1;
              UNTIL (flo_sent > 0.001) OR (cn>400);
              WRITELN ('flag3');
100:          rk := s_tim;
              IF flo_sent > 0.001
              THEN BEGIN
                    WRITE (out2,flo_sent:6:2,'       ');
                    FOR ii:= 1 TO no DO
                    WITH arc[arcs_aray[ii]] DO
                    BEGIN
                          WRITE (out2,node1:8,rk:8:2);
                          rk := rk + dist;
                    END;
                    IF no <> 0
                    THEN WITH arc[arcs_aray[no]] DO
                        WRITELN (out2,node2:8,rk:8:2);
                  END;
        END  { while dem_nod_set <> [] }
        ELSE { mincost infeasible -------------------------------}
        WHILE (dem_nod_set <> []) AND (cn<180) DO
        BEGIN
              WRITE ('Dem nod set : ');
              FOR ii := 1 TO no_nod DO
                    IF ii IN  dem_nod_set
                    THEN WRITE (ii:2); WRITELN;
              FOR ii := 1 TO snn DO
              FOR jj := 1 TO dnn DO
              IF (assign[ii,jj] > 0) and (dmn[jj].nd IN dem_nod_set )
              THEN GOTO 350;
350:          s_nod := spn[ii].nd;
              d_nod := dmn[jj].nd;
              avail_tim := dmn[jj].tim - spn[ii].tim;
              flo_sent := smaller (spn[ii].sup, dmn[jj].dem);
              flo_sent := smaller (flo_sent, assign[ii,jj]);
              istore := ii; jstore := jj;
              rk := dmn[jj].tim;
              excl_arc := []; tried := FALSE;
375:          find_short_ex_aray (s_nod,d_nod,excl_arc,arcs_aray,no);
              { Here arcs_aray's storage is in reverse order }
              motion_tim := 0;
```

```
      FOR ii := 1 TO no DO
      motion_tim := motion_tim + arc[arcs_aray[ii]].dist;
      IF (motion_tim > avail_tim) AND NOT tried
      THEN BEGIN
              WRITELN ('Time infeasible');
              tried := TRUE;
              IF excl_arc <> []
              THEN BEGIN
                      excl_arc := [];
                      goto 375;
                   END;
          END;
      WRITELN ('s_nod',s_nod:3,'  d_nod',d_nod:3);
      WRITE ('Short aray');
      FOR ii := no DOWNTO 1 DO
              WRITE (arcs_aray[ii]:4);
      WRITELN;
      { We check the residual capacities of the arcs, and time feasi}
      travl_tim := 0;
      FOR ii := 1 TO no DO
      WITH arc[arcs_aray[ii]] DO
      BEGIN
              tim_dt := rk-travl_tim;
              tim_st := tim_dt - dist;
              IF tim_st < 0  THEN tim_st := 0;
380:          passed := TRUE;
              FOR tt := round(tim_dt) DOWNTO round(tim_st) DO
              BEGIN
                      IF res_cap[tt] <= 0
                      THEN BEGIN
                              tim_dt := tt - 1;
                              tim_st := tim_dt - dist;
                              passed := FALSE;
                           END
                      ELSE flo_sent := smaller(flo_sent,res_cap[tt]);
              END;
              IF NOT passed
              THEN IF degree[arc[arcs_aray[ii]].node2] > 1
                   THEN excl_arc := excl_arc + [arcs_aray[ii]]
                   ELSE tried := TRUE;
```

```
            travl_tim := rk - tim_st;
            IF NOT passed AND ((travl_tim<avail_tim) OR tried)
            THEN GOTO 380;
            slack_tim := avail_tim - travl_tim;
            IF (slack_tim < 0) AND (motion_tim<avail_tim)
                            AND NOT tried
            THEN BEGIN
                    WRITE ('Excl Arc ');
                    FOR jj := 1 TO 2*no_arc DO
                    IF jj IN excl_arc
                    THEN WRITE (jj : 4);
                    WRITELN;
                    WRITELN ('Time infeasible (temp)');
                    GOTO 375;
                 END;
            { else if motion_tim > tim then delays }
     END;

     IF avail_tim < travl_tim
     THEN BEGIN
            WRITE ('Time infeasibility  ');
            delay := travl_tim - avail_tim;
          END
     ELSE delay := 0;
     WRITELN ('delay ',delay:5:2);
     Update_delay_info(delay,flo_sent,delay_freq,wtd_delay,ptr);
     FOR ii := no DOWNTO 1 DO
     WITH arc[arcs_aray[ii]] DO
     BEGIN
            FOR tt := round(tim_st) TO round(tim_dt) DO
            BEGIN
                    res_cap[tt] := res_cap[tt] - flo_sent;
                    load[tt] := load[tt] + flo_sent;
            END;
            WRITELN ('arc no ',arcs_aray[ii]:2,
'  Res capa ',res_cap[tt]:3:0,'  load ',load[tt]:3:0);
            WRITE ('tim s t : ',tim_st:3:0);
            WRITELN (tim_dt:3:0);
            period_cost := period_cost + flo_sent* cost_g;
     END
```

```
          WRITELN ('flo sent ',flo_sent:4:1);
          assign[istore,jstore]:= assign[istore,jstore] -flo_sent;
          spn[istore].sup := spn[istore].sup - flo_sent;
          time := spn[istore].tim;
          WITH node[s_nod] DO
          BEGIN
                  dema_supl[time] := dema_supl[time] - flo_sent;
                  requ_avai[time] := requ_avai[time] - flo_sent;
                  cum_dem_sup_g := cum_dem_sup_g - flo_sent;
          END;
          dmn[jstore].dem := dmn[jstore].dem - flo_sent;
          time := dmn[jstore].tim;
          WITH node[d_nod] DO
          BEGIN
                  dema_supl[time] := dema_supl[time] + flo_sent;
                  requ_avai[time] := requ_avai[time] + flo_sent;
                  cum_dem_sup_g := cum_dem_sup_g + flo_sent;
                  IF cum_dem_sup_g > -0.001
                  THEN dem_nod_set := dem_nod_set - [d_nod];
          END;
          IF flo_sent > 0.001
          THEN BEGIN
                  WRITE (out2,flo_sent:6:2,'       ');
                  FOR ii := no DOWNTO 1 DO
                  WITH arc[arcs_aray[ii]] DO
                          WRITE (out2,node1:8,tim_st:8:2);
                  IF no <> 0
                  THEN WITH arc[arcs_aray[1]] DO
                       WRITELN(out2,node2:8,(tim_st+dist):8:2)
                  ELSE WRITELN (out2,s_nod:8,spn[istore].tim:6,d_nod:8,
                                   dmn[jstore].tim:6);
               END;
          cn := cn +1;

      END; { while (dem_nod_set <> []) and (cn<180) do}
      IF dem_nod_set <> [] THEN
      WRITELN ('Exiting while loop !! goods demand unsatisfied');
      WRITE (out5,period_no:4, period_cost:12:2);
      total_cost := total_cost + period_cost;

300:
```

```
{ Empty wagon scheduling
  ---------------------- }
        WRITELN; WRITELN ('Empty wagon scheduling');
        WRITELN ('Period no : ',period_no:2);
        period_cost := 0;
        dem_nod_set := [];
        FOR ii := 1 TO no_nod DO
        WITH node[ii] DO
        BEGIN
                IF period_no = 1
                THEN BEGIN
                        cum_dem_sup_w := 0;
                        FOR tt := 1 TO period_lenth DO
                        IF requ_avai[tt] > 0.001
                        THEN cum_dem_sup_w := cum_dem_sup_w + requ_avai[tt];
                     END
                ELSE { period_no > 1 }
                        FOR tt := (period_no-1)*period_lenth + 1
                                to period_no*period_lenth DO
                { Consider demand for wagons in current time period }
                        BEGIN
                                IF requ_avai[tt] < -0.001
                                THEN BEGIN
                                        cum_dem_sup_w := cum_dem_sup_w
+ requ_avai[tt];
                                        dem_nod_set := dem_nod_set + [ii];
                                     END;
                { Consider suply in wagons in previous time period }
                                IF requ_avai[tt] > 0.001
                                THEN cum_dem_sup_w := cum_dem_sup_w
+ requ_avai[tt];
                        END;
        END;
        IF period_no = 1
        THEN BEGIN
                WRITELN ('Empty wagon scheduling is assumed
to be done in the previous time horizon');
                WRITELN (out5);




                { Also mention the wagons for which assume this }
```

```
          GOTO 600;
       END;

WRITELN ('Node cum_dem_sup_w');
FOR ii := 1 TO no_nod DO
WRITELN (ii:4,node[ii].cum_dem_sup_w:8:1);
          WRITE ('Dem nod set : ');
          FOR ii := 1 TO no_nod DO
          IF ii IN  dem_nod_set
          THEN WRITE (ii:2); WRITELN;
dnn := 0; { No.of dem nodes }
snn := 0; { No.of sup nodes }
FOR ii := 1 TO no_nod+1 DO
BEGIN
          ddeg[ii] := 0;
          sdeg[ii] := 0;
END;
FOR ii := 1 TO no_nod DO
WITH node[ii] DO
BEGIN
          FOR tt := period_lenth+1 TO T_ DO
          IF requ_avai[tt] < -0.001
          THEN BEGIN
                    dnn := dnn + 1;
                    ddeg[ii] := ddeg[ii] + 1;
                    WITH dmn[dnn] DO
                    BEGIN     nd := ii;
                              dem := abs(requ_avai[tt]);
                              tim := tt;
                    END;
               END
END;
FOR ii := 1 TO no_nod DO
WITH node[ii] DO
BEGIN
          FOR tt := 1 TO T_ DO
          IF requ_avai[tt] > 0.001
          THEN BEGIN




                    snn := snn + 1;
                    sdeg[ii] := sdeg[ii] + 1;
          WRITE (dmn[ii].dem:3:0);
          WRITELN;
     WRITE ('           ');
     FOR ii := 1 TO dnn DO
          WRITE (dmn[ii].tim:3);
          WRITELN;
     FOR s_nod := 1 TO no_nod DO
     FOR jj:= spt[s_nod] TO spt[s_nod+1]-1 DO
     BEGIN
          WRITE (spn[jj].nd:3, spn[jj].sup:3:0,spn[jj].tim:3);
          FOR cur_nod := 1 TO no_nod DO
          FOR kk:= dpt[cur_nod] TO dpt[cur_nod+1]-1 DO
          WRITE (assign[jj,kk]:3:0);
          {WRITE (shortest_dist(spn[jj].nd,dmn[kk].nd):3:0);}
          WRITELN;
     END;

     WRITELN (out2,'Empty Wagon Movement      Period no : ',period_no:2);
     WRITELN (out2,'Flow     ----     Node     Time     ----');
     cn := 1; s_nod := 1;
     WHILE (dem_nod_set <> []) AND (cn<200) DO
     BEGIN
          WRITE ('Dem nod set : ');
          FOR ii := 1 TO no_nod DO
               IF ii IN  dem_nod_set
               THEN WRITE (ii:2); WRITELN;
          FOR ii := 1 TO snn DO
          FOR jj := 1 TO dnn DO
               IF assign[ii,jj] > 0.001
               THEN GOTO 550;
550:      s_nod := spn[ii].nd;
          d_nod := dmn[jj].nd;
          avail_tim := dmn[jj].tim - spn[ii].tim;
          flo_sent := smaller (spn[ii].sup, dmn[jj].dem);
          flo_sent := smaller (flo_sent, assign[ii,jj]);
          istore := ii; jstore := jj;
          rk := spn[ii].tim;
          excl_arc := []; tried := FALSE;
575:      find_short_ex_aray (s_nod,d_nod,excl_arc,arcs_aray,no);



          { Here arcs_aray's storage is in reverse order }
```

```
            motion_tim := 0;
            FOR ii := 1 TO no DO
            motion_tim := motion_tim + arc[arcs_aray[ii]].dist;
            IF (motion_tim > avail_tim) AND NOT tried
            THEN BEGIN
                    WRITELN ('Time infeasible');
                    tried := TRUE;
                    IF excl_arc <> []
                    THEN BEGIN
                            excl_arc := [];
                            goto 575;
                         END;
                 END;
            WRITELN ('s_nod',s_nod:3,'  d_nod',d_nod:3);
            WRITE ('Short aray');
            FOR ii := no DOWNTO 1 DO
                    WRITE (arcs_aray[ii]:4);
            WRITELN;
            { We check the residual capacities of the arcs,
and time feasibility before sending the flo }
            travl_tim := 0;
            FOR ii := no DOWNTO 1 DO
            WITH arc[arcs_aray[ii]] DO
            BEGIN
                    tim_st := rk+travl_tim;
580:                passed := TRUE;
                    FOR tt := round(tim_st) TO round(tim_st+dist) DO
                    BEGIN
                            IF res_cap[tt] <= 0
                            THEN BEGIN
                                    tim_st := tt + 1;
                                    passed := FALSE;
                                 END
                            ELSE flo_sent := smaller(flo_sent,res_cap[tt]);
                    END;
                    IF NOT passed
                    THEN excl_arc := excl_arc + [arcs_aray[ii]];
                    travl_tim := (tim_st-rk) + dist;
                    IF NOT passed AND ((travl_tim<avail_tim) OR tried)




                    THEN GOTO 580;
                    slack_tim := avail_tim - travl_tim;
                    IF (slack_tim < 0) AND (motion_tim<avail_tim)
AND NOT tried
                    THEN BEGIN
                          WRITELN ('Time infeasible (temp)');
                          GOTO 575;
                        END;
                    { else if motion_tim > tim then delays }
              END;

              IF avail_tim < travl_tim
              THEN BEGIN
                    WRITE ('Time infeasibility  ');
                    delay := travl_tim - avail_tim;
                  END
              ELSE delay := 0;
              WRITELN ('delay ',delay:5:2);
              Update_delay_info(delay,flo_sent,delay_freq,wtd_delay,ptr);
              FOR ii := no DOWNTO 1 DO
              WITH arc[arcs_aray[ii]] DO
              BEGIN
                    FOR tt := round(tim_st) TO round(tim_st+dist) DO
                    BEGIN
                          res_cap[tt] := res_cap[tt] - flo_sent;
                          load[tt] := load[tt] + flo_sent;
                    END;
                    WRITELN ('arc no ',arcs_aray[ii]:2,
'  Res capa ',res_cap[tt]:3:0,'  load ',load[tt]:3:0);
                    WRITE ('tim s t : ',tim_st:3:0);
                    tim_dt  := tim_st + dist;
                    WRITELN (tim_dt:3:0);
                    period_cost := period_cost + flo_sent*cost_w;
              END;
              WRITELN ('flo sent ',flo_sent:3:0);
              assign[istore,jstore]:= assign[istore,jstore] -flo_sent;
              spn[istore].sup := spn[istore].sup - flo_sent;
              time := spn[istore].tim;
              WITH node[s_nod] DO
              BEGIN




                    requ_avai[time] := requ_avai[time] - flo_sent;
                    cum_dem_sup_w := cum_dem_sup_w - flo_sent;
              END;
              dmn[jstore].dem := dmn[jstore].dem - flo_sent;
              time := dmn[jstore].tim;
              WITH node[d_nod] DO
              BEGIN
                    requ_avai[time] := requ_avai[time] + flo_sent;
                    cum_dem_sup_w := cum_dem_sup_w + flo_sent;
                    IF cum_dem_sup_w > -0.001
                    THEN dem_nod_set := dem_nod_set - [d_nod];
              END;
              IF flo_sent > 0.001
              THEN BEGIN
                    WRITE (out2,flo_sent:6:2,'       ');
                    FOR ii := no DOWNTO 1 DO
                    WITH arc[arcs_aray[ii]] DO
                          WRITE (out2,node1:8,tim_st:8:2);
                    IF no <> 0
                    THEN WITH arc[arcs_aray[1]] DO
                        WRITELN(out2,node2:8,(tim_st+dist):8:2)
                    ELSE WRITELN (out2,s_nod:8,spn[istore].tim:6,
d_nod:8,dmn[jstore].tim:6);
                  END;
              cn := cn +1;

        END; { while (dem_nod_set <> []) and (cn<200) do}
        IF dem_nod_set <> []
        THEN WRITELN ('Exiting while loop !! emp wag dem unsatisfied');
        WRITELN (out5,period_cost:12:2);
        total_cost := total_cost + period_cost;
600:    IF period_no* period_lenth < T_hor
        THEN BEGIN
              period_no := period_no + 1;
              GOTO 50;
            END;
        REWRITE (out1,'outdelay.dat');
        WRITELN (out1,'Delay    Frequency  % Delay   wtd % Delay');
        cumsum := 0;  tot_delay := 0;
        FOR ii := 0 TO ptr DO




        BEGIN
```

```
            cumsum := cumsum + delay_freq[ii];
            tot_delay := tot_delay + wtd_delay[ii];
        END;
        FOR ii := 0 TO ptr DO
        WRITELN (out1,ii:4,'     ',delay_freq[ii]:4,delay_freq[ii]/
cumsum*100:12:2,
wtd_delay[ii]/tot_delay*100:12:2);
        WRITELN (out1);
        WRITELN (out1,'Total Frequency ',cumsum:4);
        WRITELN (out1,'Total Wtd delay ',tot_delay:6:2);
        WRITELN (out1,'Average Delay   ',tot_delay/cumsum:6:2);
        CLOSE (out1);
        CLOSE (out2);
        REWRITE (out3,'outload.dat');
        WRITELN (out3,'Arc no    Load     Time');
        FOR ii := 1 TO 2*no_arc DO
        WITH arc[ii] DO
        BEGIN
            maxload := 0;
            FOR tt := 1 TO T_hor DO
            BEGIN
                IF maxload < load[tt]
                THEN BEGIN
                    maxload := load[tt];
                    time := tt;
                  END;
            END;
            WRITELN (out3,ii:6,maxload:8:0,time:8);
        END;
        CLOSE (out3);
        REWRITE(out4,'dailyload.d');
        period_no := 1;
        WHILE  period_no * period_lenth <= T_hor  DO
        BEGIN
            WRITELN (out4,'Period_no : ',period_no:2);
            WRITELN (out4,'Arc No.         Time');
            WRITE (out4,'         ');
            FOR tt := (period_no-1)*period_lenth + 1  to
                        period_no * period_lenth  DO



                        WRITE (out4,tt:4);
```

```
                              WRITELN (out4);
              FOR ii := 1 TO 2 * no_arc DO
              WITH arc[ii] DO
              BEGIN
                     WRITE (out4,ii:4,'    ');
                     FOR tt := (period_no-1)*period_lenth + 1  to
                                   period_no * period_lenth  DO
                     WRITE (out4,load[tt]:4:1);
                     WRITELN (out4);
              END;
              period_no := period_no + 1;
       END;
       CLOSE (out4);
       WRITELN (out5);
       WRITELN (out5,'Total Cost : ',total_cost:12:2);
       CLOSE (out5);
END.
```

APPENDIX C

PROBLEMS SOLVED AND THEIR RESULTS

Here we present results of the 6 more problems. All the problems solved, took a CPU time in the range of 27-32 seconds of HP 9000. The results obtained are more less similar to those presented in the chapter 5. However here we have noticed that for the problems having tighter capacities of the arcs have not only more delays but also more costs of transportation, which appears clearly in graphs.

PROBLEMS 3 AND 4

The problems 1 & 2 are solved as 5 (instead of 4) period problems. Time horizon = 96 time units, Period Length = 20 time units

FINAL RESULTS FOR THE PROBLEMS 3 AND 4

Table C.1 Final Results for the Problem 3 and 4

	Empty Wagons	Transportation Costs	Delays
Problem 3	128	36,446	4.26
	153	35,968	3.97
	178	35,565	3.85
	203	35,105	3.66
Problem 4	128	39,677	4.43
	153	38,570	4.12
	178	38,261	3.94
	203	37,547	3.73

Transport. Cost Vs. Wagons Available

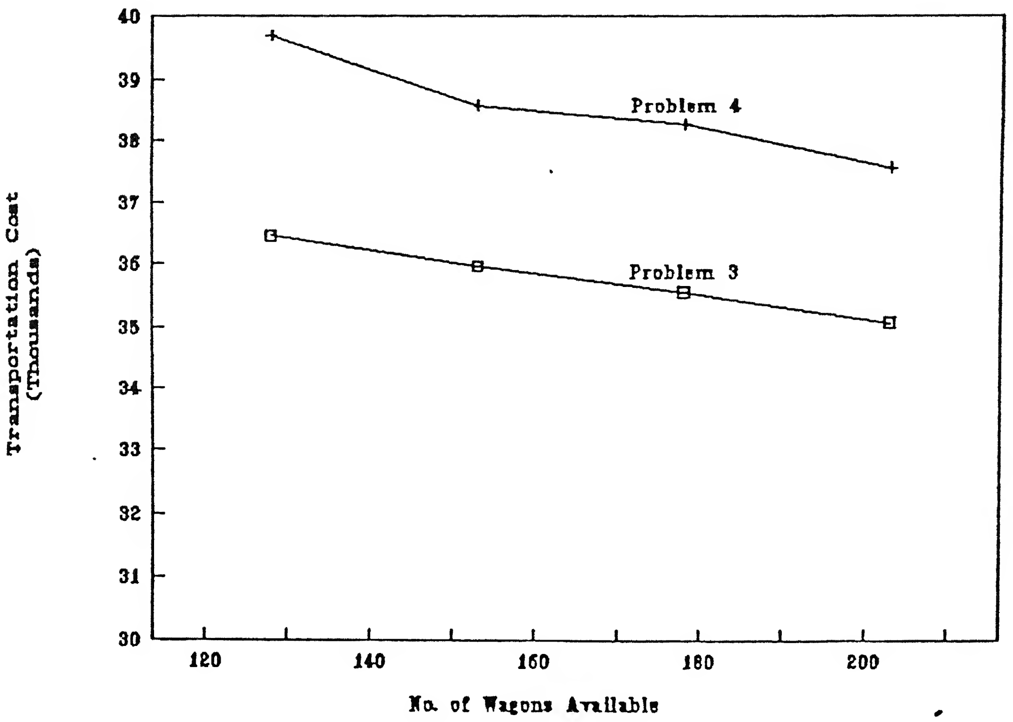


FIG. C.1

Delays Vs. No. of Wagons Available

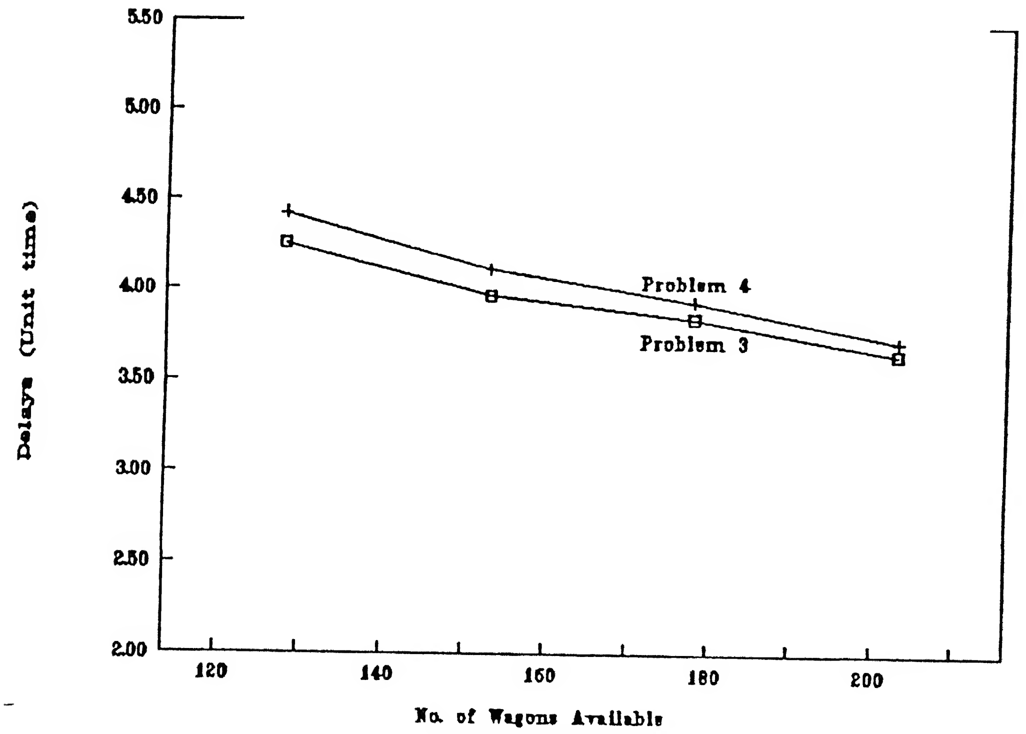


FIG. C.2

INPUT FOR THE PROBLEMS 5 AND 6

Table C.2.1 Supply at various nodes

Supply Node		Supply, time (day)
3	Kanpur	4,3; 6,4; 7,5; 5,6; 10,7; 5,9; 4,10;
5	Calcutta	6,1; 7,4; 12,5; 5,6; 2,11; 8,12;
7	Vishakhap.	8,2; 5,3; 4,2; 6,5; 7,6; 5,9; 6,10;
8	Madras	5,1; 4,2; 8,7; 5,8; 4,11; 6,12;
13	Ahmadabad	2,1; 8,3; 5,7; 6,8; 6,11; 7,12;
14	Manmad	3,1; 4,2; 3,5; 4,6; 3,9; 4,10;
16	Nagpur	8,4; 5,1; 5,7; 4,8; 7,9; 5,11; 10,12
19	Bhopal	7,1; 5,2; 10,3; 4,5; 6,6; 8,9; 5,10;
20	Indore	5,2; 6,1; 2,7; 8,8; 8,9; 5,10; 4,11;

Table C.2.2 Demands at various nodes

Demand node		Demand, time (day)
2	Jhansi	4,3; 6,4; 4,5; 6,6; 8,9; 5,10;
18	Hydrabad	5,2; 6,1; 5,7; 6,8; 6,11; 7,12; 5,9; 6,10
4	Lucknow	8,1; 7,4; 8,3; 6,5; 7,6; 8,7; 5,8; 4,11; 6,12; 3,9; 4,10;
9	Bangalore	8,4; 5,1; 4,2; 3,5; 4,6; 7,9; 5,11; 10,12
6	Jamshedpur	3,1; 4,2; 5,7; 4,8; 2,11; 8,12
1	Delhi	5,1; 4,2; 8,5; 5,6; 4,5; 2,7; 8,8; 8,9 5,10; 4,11
12	Bombay	8,2; 15,3; 7,5; 5,6; 10,7; 5,9; 4,10

The time horizon = 12 days, and period length = 4 day

Table C.2.3 Table for capacities of the arcs.

Arc No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Pr. 5	17	4	13	5	23	32	7	17	7	7	9	10	13	5	6	5	6	7	10	3	21	6	21	22	13	5	8	5	12
Pr. 6	14	3	12	5	20	30	7	15	7	7	9	10	15	5	6	5	6	7	10	3	21	6	21	22	11	5	8	5	11

FINAL RESULTS FOR PROBLEMS 5 AND 6

Table C.2.4 Final Results for the Problem 5

Empty Wagons	Transportation Costs	Delays
205	22,471	4.36
230	21,368	4.28
255	21,042	4.15
280	21,042	4.17

Table C.2.5 Final Results for the Problem 6

Empty Wagons	Transportation Costs	Delays
205	22,393	5.64
230	21,480	5.64
255	21,153	5.33
280	21,153	5.32

Transport. Cost Vs. Wagons Available

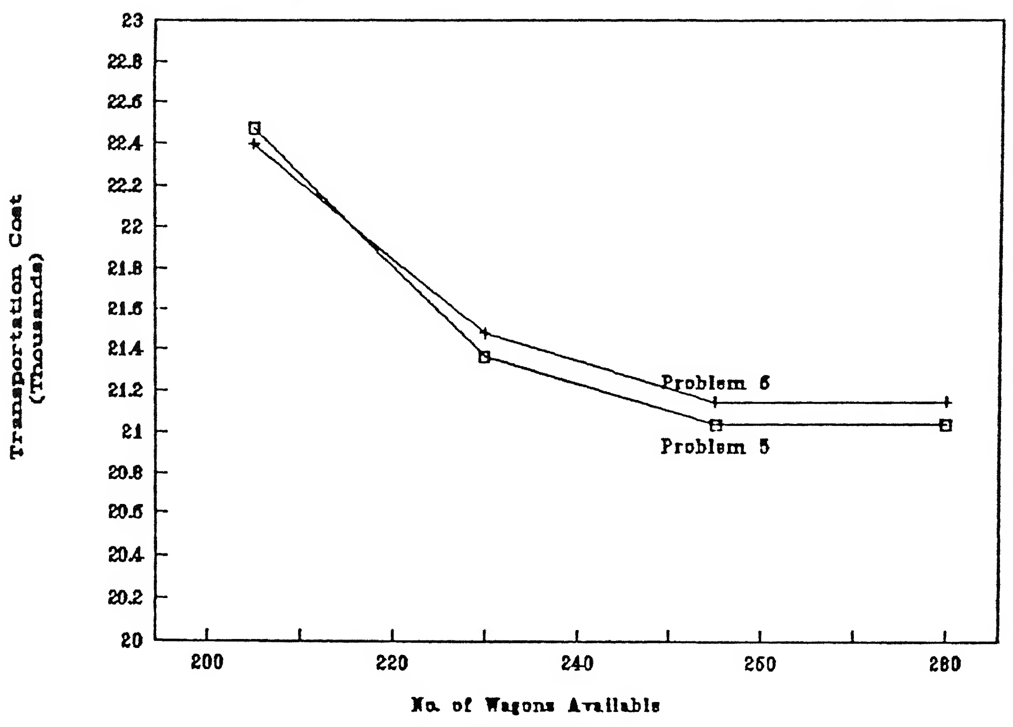


FIG. C.3

Delays Vs. No. of Wagons Available

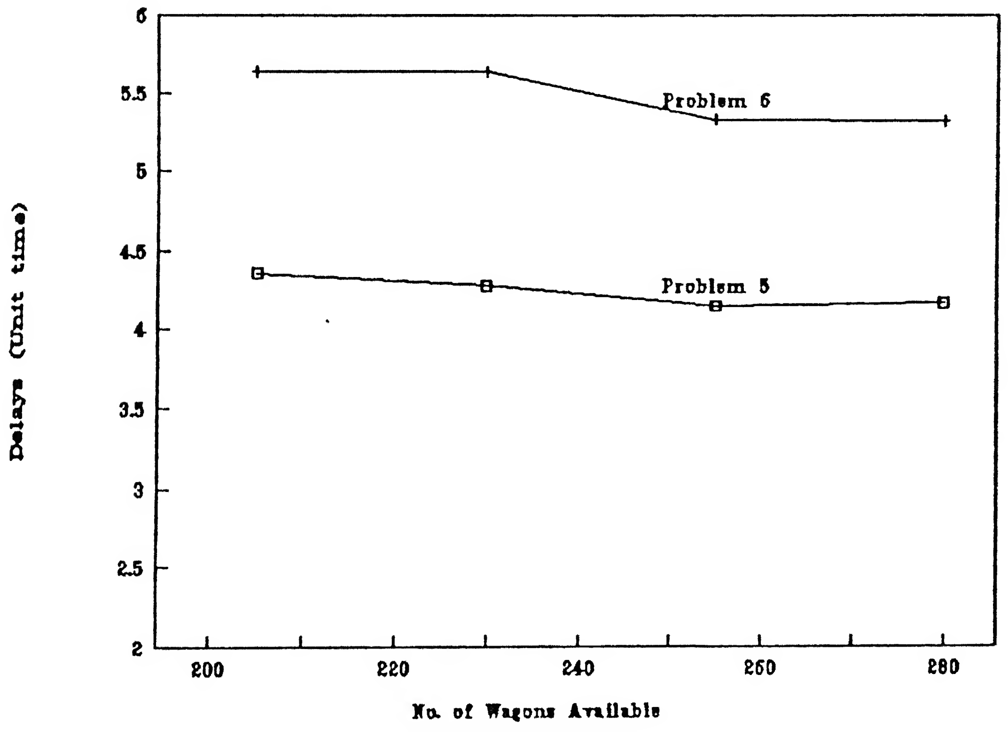


FIG. C.4

INPUT FOR THE PROBLEMS 5 AND 6

Table C.2.1 Supply at various nodes

Supply Node		Supply, time (day)
3	Kanpur	4,3; 6,4; 7,5; 5,6; 10,7; 5,9; 4,10;
5	Calcutta	6,1; 7,4; 12,5; 5,6; 2,11; 8,12;
7	Vishakhap.	8,2; 5,3; 4,2; 6,5; 7,6; 5,9; 6,10;
8	Madras	5,1; 4,2; 8,7; 5,8; 4,11; 6,12;
13	Ahmadabad	2,1; 8,3; 5,7; 6,8; 6,11; 7,12;
14	Manmad	3,1; 4,2; 3,5; 4,6; 3,9; 4,10;
16	Nagpur	8,4; 5,1; 5,7; 4,8; 7,9; 5,11; 10,12
19	Bhopal	7,1; 5,2; 10,3; 4,5; 6,6; 8,9; 5,10;
20	Indore	5,2; 6,1; 2,7; 8,8; 8,9; 5,10; 4,11;

Table C.2.2 Demands at various nodes

Demand node		Demand, time (day)
2	Jhansi	4,3; 6,4; 4,5; 6,6; 8,9; 5,10;
18	Hydrabad	5,2; 6,1; 5,7; 6,8; 6,11; 7,12; 5,9; 6,10
4	Lucknow	8,1; 7,4; 8,3; 6,5; 7,6; 8,7; 5,8; 4,11; 6,12; 3,9; 4,10;
9	Bangalore	8,4; 5,1; 4,2; 3,5; 4,6; 7,9; 5,11; 10,12
6	Jamshedpur	3,1; 4,2; 5,7; 4,8; 2,11; 8,12
1	Delhi	5,1; 4,2; 8,5; 5,6; 4,5; 2,7; 8,8; 8,9 5,10; 4,11
12	Bombay	8,2; 15,3; 7,5; 5,6; 10,7; 5,9; 4,10

The time horizon = 12 days, and period length = 4 day

Table C.2.3 Table for capacities of the arcs.

Arc No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Pr. 5	17	4	13	5	23	32	7	17	7	7	9	10	13	5	6	5	6	7	10	3	21	6	21	22	13	5	8	5	12
Pr. 6	14	3	12	5	20	30	7	15	7	7	9	10	15	5	6	5	6	7	10	3	21	6	21	22	11	5	8	5	11

FINAL RESULTS FOR PROBLEMS 5 AND 6

Table C.2.4 Final Results for the Problem 5

Empty Wagons	Transportation Costs	Delays
205	22,471	4.36
230	21,368	4.28
255	21,042	4.15
280	21,042	4.17

Table C.2.5 Final Results for the Problem 6

Empty Wagons	Transportation Costs	Delays
205	22,393	5.64
230	21,480	5.64
255	21,153	5.33
280	21,153	5.32

Transport. Cost Vs. Wagons Available

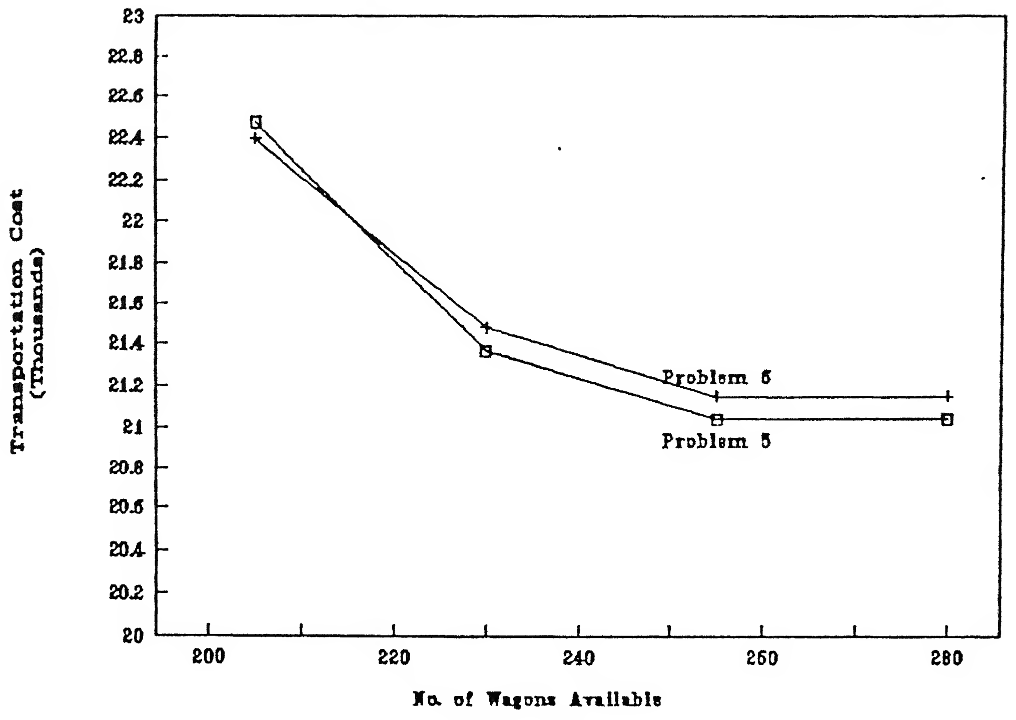


FIG. C.3

Delays Vs. No. of Wagons Available

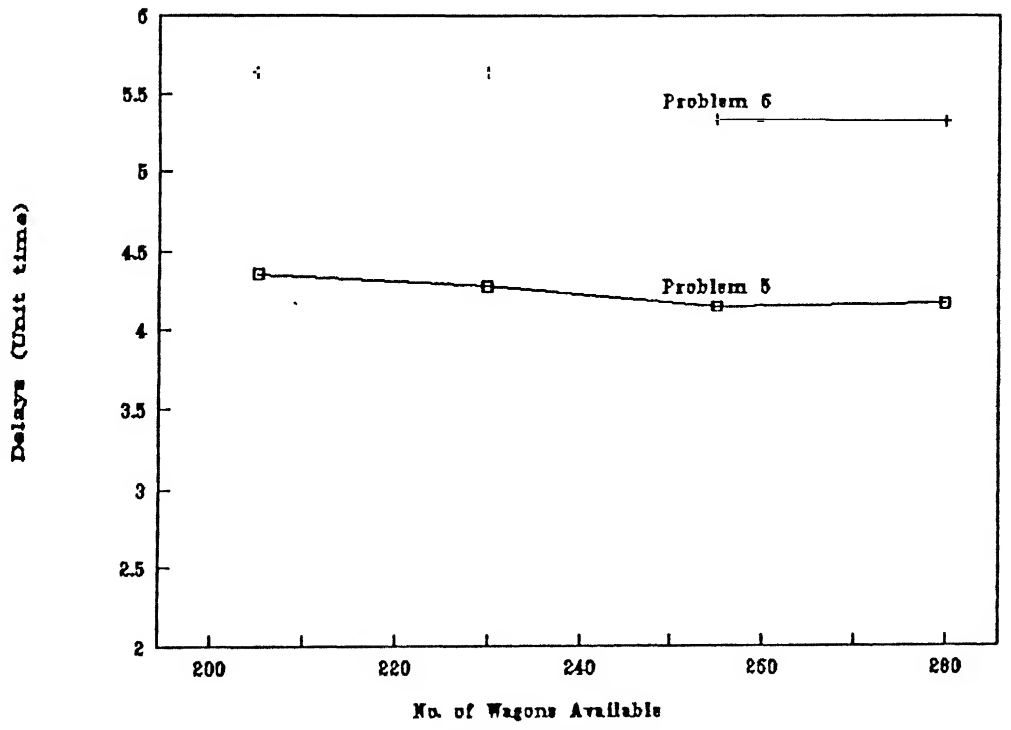


FIG. C.4

INPUT FOR THE PROBLEMS 7 AND 8

Table C.3.1 Supplies at various nodes

Supply Node		Supply, time (day)
15	Bhusaval	6,1; 7,2; 1,6; 4,7; 6,11; 7,12
20	Indore	8,3; 5,4; 8,8; 7,9; 5,10; 8,12; 5,13
3	Kanpur	4,5; 5,1; 4,6; 3,7; 4,8; 1,14; 4,15
4	Lucknow	4,2; 2,3; 4,9; 2,10; 8,11; 7,12; 5,13
9	Bangalore	4,4; 3,5; 4,1; 4,6; 5,7; 4,14; 3,16; 4,11
6	Jamshedpur	8,2; 7,3; 5,4; 8,8; 5,9; 4,12; 2,13
1	Delhi	1,5; 4,1; 6,10; 7,6; 4,14; 5,15

Table C.3.2 Demands at various node

Demand Node		Demand, time (day)
19	Bhopal	6,2; 4,10; 4,6; 5,15; 7,11
13	Ahmdabad	1,5; 4,1; 3,8; 5,9; 4,13; 4,14
7	Vishakha.	6,4; 1,10; 2,6; 5,7; 3,11; 5,12
11	Pune	1,2; 2,3; 3,7; 4,8; 1,9; 2,14; 6,15
17	Balharsha	7,5; 2,1; 5,10; 3,6; 8,13
10	Daund	1,4; 8,9; 5,11; 3,12
14	Manmad	3,2; 3,3; 2,7; 6,8; 3,13; 1,15
16	Nagpur	4,5; 5,1; 3,10; 5,6; 1,15; 2,11; 5,12
2	Jhansi	1,3; 7,4; 4,8; 4,9; 3,13; 9,14
18	Hydrabad	3,1; 4,2; 5,6; 7,7; 4,11; 4,12

The time horizon = 15 days and period length = 5 days.

Table C.3.3 Table for capacities of the arcs.

Arc No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Pr. 7	19	5	25	8	20	24	5	21	6	18	18	7	6	23	3	14	5	16	8	5	5	8	8	4	5	6	7	8	2
Pr. 8	16	5	20	8	20	20	5	20	6	15	10	6	7	22	3	12	5	15	8	5	5	8	8	5	5	6	6	8	2

FINAL RESULTS FOR THE PROBLEMS 7 AND 8

Table C.3.4 Final Results for problem 7

Empty Wagons	Transportation Costs	Delay
157	21,753	5.25
182	20,790	5.03
207	19,786	4.61
232	17,843	4.62

Table C.3.4 Final Results for problem 8

Empty Wagons	Transportation Costs	Delay
157	22,187	6.06
182	21,241	5.63
207	20,223	5.32
232	18,300	4.72

Transport. Cost Vs. Wagons Available

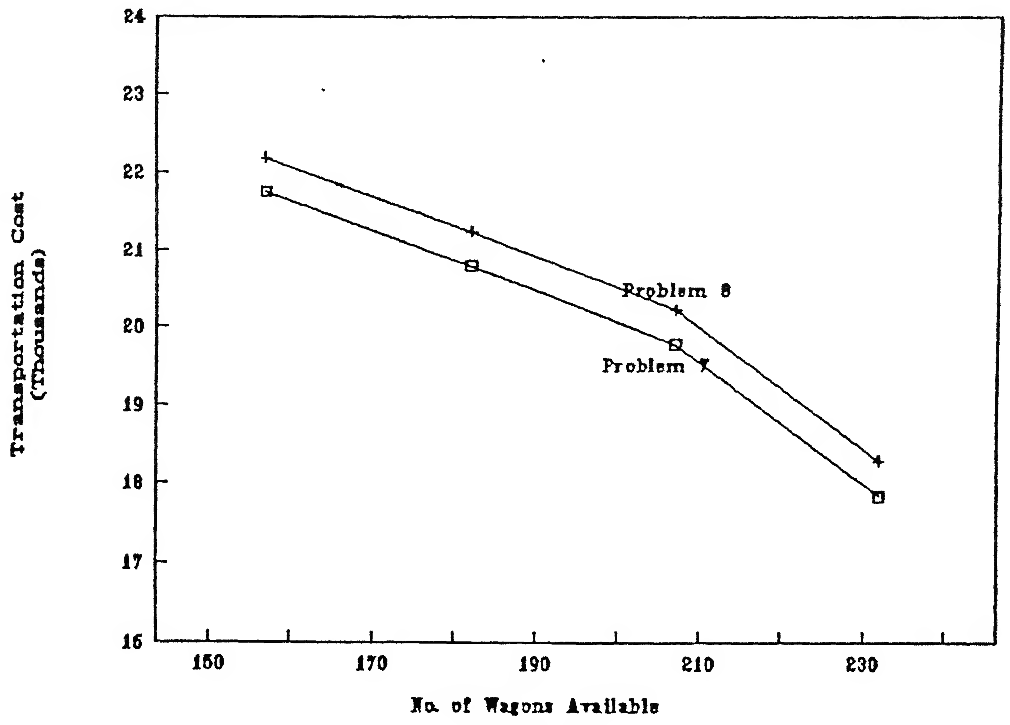


FIG. C.5

Delays Vs. No. of Wagons Available

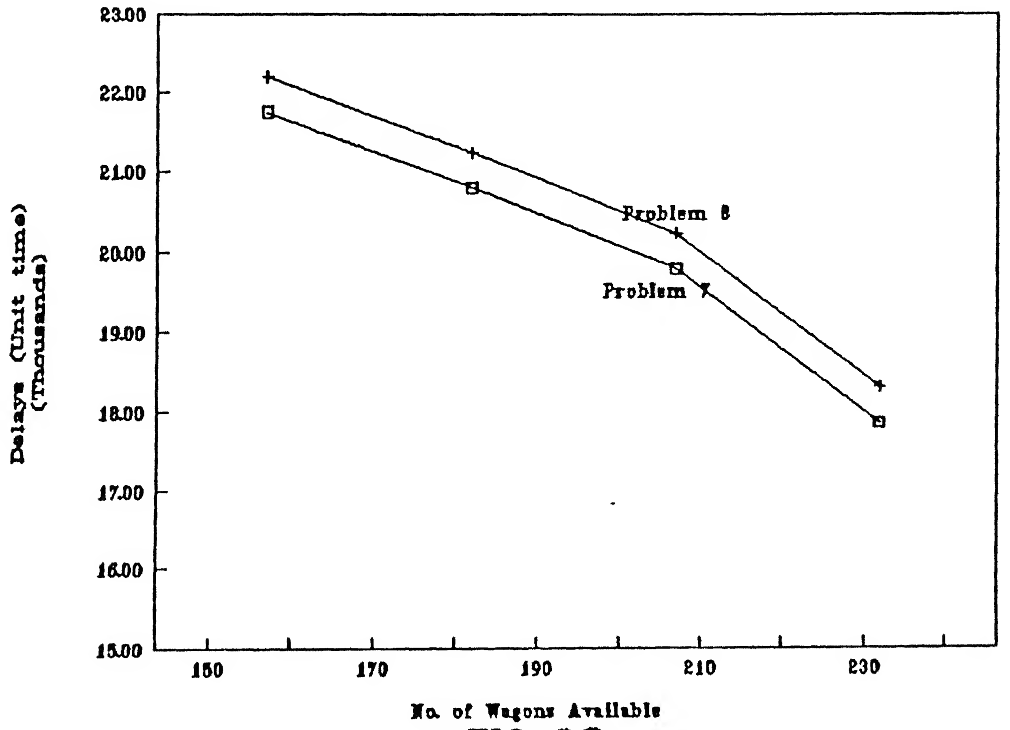


FIG. C.6

DISCUSSION

Here we give minutes of discussion during the thesis presentation.

Q. What is the smallest unit of time for which scheduling is done ? What is the basis of deciding the its level ?

Ans. We have used 1 unit time = 4 hrs. in our scheduling problem. The criteria for deciding the smallest unit of time are
1. Travelling time for arc with smallest dist 2. Smaller the unit time finer the level of scheduling and more computational time.

Q. Where do we use simulation in the methodology ?

Ans. During routing we try incorporate to have essence of real life logic and relationships. Further, we perform experiments we the model developed so as to decide what fraction of capacity of arcs per period should be taken as the capacity of arc per unit time.

Q. What is alternative methodology that can be used ?

Ans. One can try to use Lagrangian Relaxation by relaxing some constraints, dualizing them and exploit some standard structure out of the problem. Also some other ways of dividing the problem into smaller subproblems, methods like Benders' Decomposition may be tried. Also one can have analogy between our problem and job shop scheduling problem. Wagons are analogous to the jobs. Wagons moving over arcs is similar to the jobs loaded on the machines. Travelling time being analogous to processing time of jobs. Delivery dates for the goods are similar to due dates of the jobs. However it should be noted that scheduling many jobs on more machines (more than 3) is itself a tough problem to solve.

Q. How the costs are expressed? Can they be expressed in terms of cost of moving a wagon per arc?

Ans. Costs are expressed as cost of moving a train of wagons per arc. It can be expressed in terms of wagons also for going into finer details of scheduling. It will involve some component of cost which is fixed and some component which is proportional to the number of wagons in train.

Q. Is it necessary to take cost proportional to the distance ?

Ans. It is not necessary to take cost proportional to the distance. In real world also cost may vary not only with distance but also depending on whether the route is electrified or uses diesel or coal engines. The algorithm will work such input also without any further modification.